

**21st International Conference in Central Europe
on
Computer Graphics, Visualization and Computer Vision**

in co-operation with

EUROGRAPHICS Association

WSCG 2013

Communication Papers Proceedings

Edited by

Vaclav Skala, University of West Bohemia, Czech Republic

**21st International Conference in Central Europe
on
Computer Graphics, Visualization and Computer Vision**

in co-operation with

EUROGRAPHICS Association

WSCG 2013

Communication Papers Proceedings

Edited by

Vaclav Skala, University of West Bohemia, Czech Republic

Vaclav Skala – Union Agency

WSCG 2013 - Communication Papers Proceedings

Editor: Vaclav Skala
c/o University of West Bohemia, Univerzitni 8
CZ 306 14 Plzen
Czech Republic
skala@kiv.zcu.cz <http://www.VaclavSkala.eu>

Managing Editor: Vaclav Skala

Published and printed by:
Vaclav Skala – Union Agency
Na Mazinách 9
CZ 322 00 Plzen
Czech Republic <http://www.UnionAgency.eu>

Hardcopy: ISBN 978-80-86943-75-6

WSCG 2013

International Program Committee

Benes, Bedrich (United States)
Benger, Werner (United States)
Bengtsson, Ewert (Sweden)
Bilbao, Javier,J. (Spain)
Biri, Venceslas (France)
Bittner, Jiri (Czech Republic)
Buehler, Katja (Austria)
Coquillart, Sabine (France)
Daniel, Marc (France)
de Geus, Klaus (Brazil)
de Oliveira Neto, Manuel Menezes (Brazil)
Debelov, Victor (Russia)
Feito, Francisco (Spain)
Ferguson, Stuart (United Kingdom)
Gain, James (South Africa)
Gudukbay, Ugur (Turkey)
Guthe, Michael (Germany)
Herout, Adam (Czech Republic)
Choi, Sunghee (Korea)
Chover, Miguel (Spain)
Chrysanthou, Yiorgos (Cyprus)
Juan, M.-Carmen (Spain)
Kim, HyungSeok (Korea)
Klosowski, James (United States)
Max, Nelson (United States)
Molla, Ramon (Spain)
Muller, Heinrich (Germany)
Murtagh, Fionn (United Kingdom)

Pan, Rongjiang (China)
Paquette, Eric (Canada)
Patow, Gustavo (Spain)
Pedrini, Helio (Brazil)
Platis, Nikos (Greece)
Reshetov, Alexander (United States)
Richardson, John (United States)
Rojas-Sola, Jose Ignacio (Spain)
Santos, Luis Paulo (Portugal)
Savchenko, Vladimir (Japan)
Skala, Vaclav (Czech Republic)
Slavik, Pavel (Czech Republic)
Sochor, Jiri (Czech Republic)
Sourin, Alexei (Singapore)
Sousa, A.Augusto (Portugal)
Sramek, Milos (Austria)
Stroud, Ian (Switzerland)
Szecsi, Laszlo (Hungary)
Teschner, Matthias (Germany)
Theussl, Thomas (Saudi Arabia)
Tokuta, Alade (United States)
Vitulano, Domenico (Italy)
Wu, Shin-Ting (Brazil)
Wuensche, Burkhard,C. (New Zealand)
Wuethrich, Charles (Germany)
Zara, Jiri (Czech Republic)
Zemcik, Pavel (Czech Republic)
Zitova, Barbara (Czech Republic)

WSCG 2013

Board of Reviewers

Agathos, Alexander	Fuenfzig, Christoph	Kurt, Murat
Assarsson, Ulf	Gain, James	Kyratzi, Sofia
Ayala, Dolors	Galo, Mauricio	Larboulette, Caroline
Backfrieder, Werner	Gobron, Stephane	Lee, Jong Kwan Jake
Barbosa, Joao	Grau, Sergi	Liu, Damon Shing-Min
Barthe, Loic	Gudukbay, Ugur	Lopes, Adriano
Battiato, Sebastiano	Guthe, Michael	Loscos, Celine
Benes, Bedrich	Hansford, Dianne	Lutteroth, Christof
Benger, Werner	Haro, Antonio	Maciel, Anderson
Bilbao, Javier,J.	Hasler, Nils	Mandl, Thomas
Biri, Venceslas	Hast, Anders	Manzke, Michael
Birra, Fernando	Hernandez, Benjamin	Marras, Stefano
Bittner, Jiri	Hernandez, Ruben Jesus Garcia	Masia, Belen
Bosch, Carles	Herout, Adam	Masood, Syed Zain
Bourdin, Jean-Jacques	Herrera, Tomas Lay	Max, Nelson
Brun, Anders	Hicks, Yulia	Melendez, Francho
Bruni, Vittoria	Hildenbrand, Dietmar	Meng, Weiliang
Buehler, Katja	Hinkenjann, Andre	Mestre, Daniel,R.
Bulo, Samuel Rota	Chaine, Raphaelle	Metodiev, Nikolay Metodiev
Cakmak, Hueseyin	Choi, Sunghee	Meyer, Alexandre
Camahort, Emilio	Chover, Miguel	Molina Masso, Jose Pascual
Casciola, Giulio	Chrysanthou, Yiorgos	Molla, Ramon
Cline, David	Chuang, Yung-Yu	Montrucchio, Bartolomeo
Coquillart, Sabine	Iglesias, Jose,A.	Morigi, Serena
Cosker, Darren	Ihrke, Ivo	Muller, Heinrich
Daniel, Marc	Iwasaki, Kei	Munoz, Adolfo
Daniels, Karen	Jato, Oliver	Murtagh, Fionn
de Geus, Klaus	Jeschke, Stefan	Okabe, Makoto
de Oliveira Neto, Manuel	Jones, Mark	Oyarzun, Cristina Laura
Menezes	Juan, M.-Carmen	Pan, Rongjiang
Debelov, Victor	Kämpe, Viktor	Papaioannou, Georgios
Drechsler, Klaus	Kanai, Takashi	Paquette, Eric
Durikovic, Roman	Kellomaki, Timo	Pasko, Galina
Eisemann, Martin	Kim, H.	Patane, Giuseppe
Erbacher, Robert	Klosowski, James	Patow, Gustavo
Feito, Francisco	Kolcun, Alexej	Pedrini, Helio
Ferguson, Stuart	Krivanek, Jaroslav	Pereira, Joao Madeiras
Fernandes, Antonio	Kurillo, Gregorij	Peters, Jorg

Pina, Jose Luis
Platis, Nikos
Post, Frits,H.
Puig, Anna
Rafferty, Karen
Renaud, Christophe
Reshetouski, Ilya
Reshetov, Alexander
Ribardiere, Mickael
Ribeiro, Roberto
Richardson, John
Rojas-Sola, Jose Ignacio
Rokita, Przemyslaw
Rudomin, Isaac
Sacco, Marco
Salveti, Ovidio
Sanna, Andrea
Santos, Luis Paulo
Sapidis, Nickolas,S.
Savchenko, Vladimir
Seipel, Stefan
Sellent, Anita

Shesh, Amit
Sik-Lanyi, Cecilia
Sintorn, Erik
Skala, Vaclav
Slavik, Pavel
Sochor, Jiri
Sourin, Alexei
Sousa, A.Augusto
Sramek, Milos
Stroud, Ian
Subsol, Gerard
Sundstedt, Veronica
Szecsi, Laszlo
Teschner, Matthias
Theussl, Thomas
Tian, Feng
Tokuta, Alade
Torrens, Francisco
Trapp, Matthias
Tytkowski, Krzysztof
Umlauf, Georg
Vasa, Libor

Vergeest, Joris
Vitulano, Domenico
Vosinakis, Spyros
Walczak, Krzysztof
WAN, Liang
Wu, Shin-Ting
Wuensch, Burkhard,C.
Wuethrich, Charles
Xin, Shi-Qing
Xu, Dongrong
Yoshizawa, Shin
Yue, Yonghao
Zalik, Borut
Zara, Jiri
Zemcik, Pavel
Zhang, Xinyu
Zhao, Qiang
Zheng, Youyi
Zitova, Barbara
Zwettler, Gerald

WSCG 2013

Communications Papers Proceedings

Contents

	Page
Rios-Soria,D., Schaeffer,S., Garza-Villarreal,S.: Hand-gesture recognition using computer-vision techniques	1
Al Hamad,H.A.: Neural-Based Segmentation Technique for Arabic Handwriting Scripts	9
Gdawiec,K.: Polynomiography and various convergence tests	15
Schiffner,D., Ritter,M., Benger,W.: Fast Normal Approximation of Point Clouds in Screen Space	21
Van Dyk,B., Lutteroth,C., Weber,G., Wuensche,B.: Using OpenGL State and History for Graphics Debugging	29
Peek,E., Wuensche,B., Lutteroth,C.: Virtual Reality Capabilities of Graphics Engines	39
Sena,D., Pereira,J., Costa,V.: Physics-based Water Interaction and Shading: the SiViFlow Algorithm	49
Nguyen,V.-S, Bac,A., Daniel,M.: Simplification of 3D Point Clouds sampled from Elevation Surfaces	60
Seib,V., Giesen,J., Grüntjens,D., Paulus,D.: Enhancing Human-Robot Interaction by a Robot Face with Facial Expressions and Synchronized Lip Movements	70
Ilgner, Kuhlmann, Eirund, Hering-Bertram: Interacting in 3D Virtual Worlds with Brain Computer Interfaces	78
Raulet,J., Boyer,V.: Comics reading: An automatic script generation	88
Murru,G., Fratarcangeli,M., Empler,T.: Practical Augmented Visualization on Handheld Devices for Cultural Heritage	97
Li,B., Mukundan,R.: Comparative Analysis of Spatial Partitioning Methods for Large-Scale, Real-time Crowd Simulation	104
Vergeest,J.S.M.: High-velocity optical flow	112
Kanzok,Th., Süß,F., Linsen,L., Rosenthal,P.: Efficient Removal of Inconsistencies in Large Multi-Scan Point Clouds	120
Liang,M., Zheng,G., Huang,X., Milledge,G., Tokuta,A.: Identification of abnormal cervical regions from colposcopy image sequences	130
Akagi,Y., Furukawa,R., Sagawa,R., Ogawara,K., Kawasaki,H.: A facial motion tracking and transfer method based on a key point detection	137

Kolcun,A.: (Semi) regular tetrahedral tilings	145
Saini,V., Gade,S., Prasad,M., Chatterjee,S.: The 3-Point Method: A Fast, Accurate and Robust Solution to Vanishing Point Estimation	151
Lacheheb,H., Aouat,S., Hamouchene,I.: MCM-CBIR: Multi Clustering Method for Content Based Image Retrieval	159
Cocias,T.T., Grigorescu,S.M., Moldoveanu,F.: Generic Fitted Primitives (GFP): Towards Full Object Volumetric Reconstruction for Service Robotics	166
Sugihara,K.: Straight Skeleton for Automatic Generation of 3-D Building Models with General Shaped Roofs	175
Alves,R.M., Sousa,L.S.R., Rodrigues,J.M.F.: PoolLiveAid: Augmented reality pool table to assist inexperienced players	184
Popov,E.V., Rotkov,S.I.: The Retrieval of NURBS-surface by Genetic Algorithm on the Basis of Point Cloud	194
Dechvijankit,A., Nagahashi,H., Aoki,K.: An Optimization of Square Parameterization	203
Borges,D., Ferreira,A.: Part-based Construction of digitized 3D objects	210
Kansal,R., Kumar,S.: A framework for detection of linear gradient filled regions and their reconstruction for vector graphics	220
Arora,N., Kumar,H., Dhaliwal,J.S., Kalra,P., Chaudhuri,P.: Improved Interactive Reshaping of Humans in Images	230
Quoc-Viet,D., Sandrine,M., Géraldine,M.: Similarity Detection for Free-Form Parametric Models	239
Diaz,R.G., Dreux,M., Coelho,L.C.G.: Generation of Parameterized Models for Vessels Design	249
Karpavičius,V., Krasauskas,R.: Real-time Visualization of Moebius Transformations in Space using Quaternionic-Bezier Approach	259

Hand-gesture recognition using computer-vision techniques

David J. Rios-Soria
 Universidad Autónoma de Nuevo
 León (UANL)
 San Nicolás de los Garza, NL, Mexico
 david.j.rios@gmail.com

Satu E. Schaeffer
 Universidad Autónoma de
 Nuevo León (UANL)
 San Nicolás de los Garza, NL,
 Mexico
 elisa.schaeffer@gmail.com

Sara E. Garza-Villarreal
 Universidad Autónoma de
 Nuevo León (UANL)
 San Nicolás de los Garza, NL,
 Mexico
 saraelena@gmail.com

ABSTRACT

We use our hands constantly to interact with things: pick them up, move them, transform their shape, or activate them in some way. In the same unconscious way, we gesticulate in communicating fundamental ideas: ‘stop’, ‘come closer’, ‘over there’, ‘no’, ‘agreed’, and so on. Gestures are thus a natural and intuitive form of both interaction and communication. Gestures and gesture recognition are terms increasingly encountered in discussions of human-computer interaction. We present a tool created for human-computer interaction based on hand gestures. The underlying algorithm utilizes only computer-vision techniques. The tool is able to recognize in real time six different hand gestures, captured using a webcam. Experiments conducted to evaluate the system performance are reported.

Keywords: Hand-gesture recognition, computer vision, human computer interaction.

1 Introduction

There are situations in which it is necessary to interact with a system without touching it. The reasons include dirty hands (when repairing a motor, for example), hygiene (to indicate the desired water temperature when washing hands in a public bathroom), and focus of attention (not wishing to redirect the sight towards the controls when operating delicate equipment or interacting with an augmented-reality scenario). The use of voice commands as an alternative to touch-based controls, such as keyboards, buttons, and touch screens, requires a quiet environment and natural language processing; voice commands are, additionally, language-specific and sensitive to dialects and to speech impediments. Another alternative is remote control through *gesture recognition*, also known as remote control “with the wave of a hand”. Common applications for this kind of control involve medical systems —provide the user sterility to avoid the spread of infections—, entertainment, and human-robot interaction [WKSE11].

The option explored in this work, *computer vision for gesture recognition*, has advantages over touch-based controls and voice commands. Our proposed hand-

gesture detection algorithm works in real time, using basic computer-vision techniques such as filters, border detection, and convex-hull detection; in addition, it only requires a standard webcam, does not need special markers on the hand, can detect the hand regardless of its position (upside down, backwards, leaned to the left or right), and is easily extended for detecting two hands at the same time.

To test this approach, user experiments were carried out and two applications that use our gesture-detection system were developed. In the first application, the detected gestures are used as commands for interaction with a GPS device; in the second one, the detected gestures control the movements of a robot.

This document is organized as follows: Section 2 discusses background for this work and Section 3 reviews related work; Section 4 presents the details of our algorithm for hand-gesture recognition, which is able to recognize six different gestures in real time. Section 5 discusses our prototype implementation and user experiments, and Section 7 offers conclusions and discusses future directions.

2 Background

The use of the hand as an input device is a method that provides natural human-computer interaction. Among the challenges of human-computer interaction is the creation of user-friendly interfaces that use natural communication. Sophisticated applications such as virtual environments or augmented-reality systems should pro-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

vide effective human-computer interaction for applications involving complex tasks. In these applications, users should be supplied with sophisticated interfaces allowing them to navigate within the system, select objects, and manipulate them.

The use of computer vision for human-computer interaction is a natural, non-intrusive, non-contact solution. Computer vision can be used for gesture detection and classification, and various approaches have been proposed to support simple applications. To recognize hand gestures using computer vision, it is first needed to detect the hand on an image or video stream. Hand detection and pose estimation involve extracting the position and orientation on the hand, fingertip locations, and finger orientation from the images. Skin-color filtering is a common method for locating the hand because of its fast implementation. Skin-color filters rely on the assumption that the hand is the only skin-colored object. Gesture classification is a research field involving many machine-learning techniques such as neural networks and hidden Markov models [SP09].

However, hand-pose estimation is still a challenge in computer vision. Several open problems remain to be solved in order to obtain robustness, accuracy, and high processing speed. The need of an inexpensive but high-speed system is rather evident. Development of these systems involves a challenge in the research of effective input/output techniques, interaction styles, and evaluation methods [EBN⁺07].

3 Related work

There are several areas where the detection of hand gestures can be used, such as device interaction, virtual-object interaction, sign-language recognition, and robot control. Wachs et al. [WKSE11] present some examples of applications such as medical assistance systems, crisis management, and human-robot interaction. In the following subsection we present some examples of gesture-based interaction systems.

3.1 Device interaction

There are works related to electronic device interaction; for example, Stergiopoulou et al. [SP09] use self-growing and self-organized neural networks for hand gesture recognition. Another example is *Finger counting* [CB03] a simple human-computer interface. Using a webcam, it interprets specific hand gestures as input to a computer system in real time.

The *UbiHand* [AM06b] is an input device that uses a miniature wrist-worn camera to track finger position, providing a natural and compact interface. A hand model is used to generate a 3D representation of the hand, and a gesture recognition system interprets finger

movements as commands. The system is a combination of a pointer position and non-chorded keystroke input device to track finger position [AM06a].

An interactive screen developed by The Alternative Agency¹ in UK is located in a department store window (Figure 1). The *Orange screen* allows interaction just by moving the hands in front of the window without the need to touch it.



Figure 1: The world's first touchless interactive shop window

Lenman et al. [LBT02] use gesture detection to interact with electronic-home devices such as televisions and DVD players.

MacLean et al. [MHP⁺01] use hand-gesture recognition for real-time teleconferencing applications. The gestures are used for controlling horizontal and vertical movement as well as zooming functions. Schlömer et al. [SPHB08] use hand-gesture recognition for interaction with navigation applications such viewing photographs on a television, whereas Roomi et al. [RPJ10] propose a hand-gesture detection system for interaction with slideshow presentations in PowerPoint. The gesture-detection system presented in Argyros et al. [AL06] allows to control remotely the computer mouse.

Sixthsense [MMC09] is a system that converts any surface into an interactive surface. In order to interact with the system, hand gesture recognition is used. In the *Sixthsense* system, color markers are used in the fingers to detect the gestures.

3.2 Virtual object interaction

Gesture detection can be used for interaction with virtual objects; there are several works that show applications for this scenario.

Hirobe et al. [HNW⁺09] have created an interface for mobile devices using image tracking. The system tracks the finger image and allows to type on an in-air keyboard and draw 3D pictures.

HandVu [Kol10] is a hand-gesture vision-based recognition system that allows interaction with virtual objects (Figure 2) *HandVu* detects the hand in a standard posture, then tracks it and recognizes key postures, all in real-time and without the need for camera or user calibration. Although easy to understand, the used gestures are not natural.

¹ <http://www.thealternative.co.uk/>

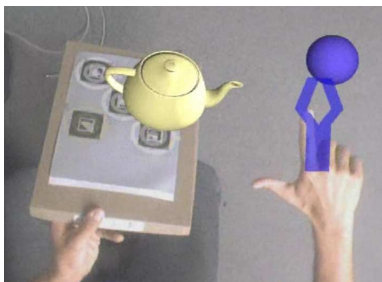


Figure 2: The gestures used in the *Handvu* system [Kol10] are not natural gestures.

Wobbrock et al. [WMW09] propose a series of gestures in order to make easier the use of interactive surfaces. Wachs et al. [WSE⁺06] use real-time hand gestures for object and window manipulation in a medical data visualization environment.

3.3 Sign language recognition

Zahedi et al. [ZM11] create a system for sign language recognition based on computer vision. Wang et al. [WP09] present a work where hand gesture detection is used in three applications: animated character interaction, virtual object manipulation, and sign language recognition.

3.4 Robot-control

Malima et al. [ÇMÖ06] use hand-gesture detection for remote robot-control. They have noted that images taken under insufficient light (especially using the webcam) have led to incorrect results. In these cases the failure mainly stems from the erroneous segmentation of some background portions as the hand region.

4 Theory

Our proposed algorithm performs hand-gesture recognition by utilizing computer-vision techniques and is able to recognize six different gestures in real-time. The processing steps included in the algorithm are explained in detail in the following subsections.

4.1 Hand recognition

Hand-recognition systems are based on the processing of an incoming digital image, preferably in real time. The first task is to separate the image of a hand from the background. This can be achieved in several ways and depends on whether the image includes only a hand against a background or the entire person. Options for detecting the hand against a background, which is the typical case for the augmented-reality setting, where the user wears a headset with a camera pointing towards

his or her field of vision, include either comparing the subsequent frames in a video (to detect movement — sensitive to motion in the background as well as shaking of the hand itself—) or using a *skin-color filter* (to classify the pixels of the image into two classes, “hand” or “background”, depending on their color values). In this work, we employ the latter approach, which is somewhat sensible to high variations of skin color (the problematic cases being very pale and very dark-skinned users). This can be done on a single frame, that is, a still photograph, but can often be improved by averaging over a few adjacent video frames; in our work we use the average over ten frames.

The skin-color filtering in such does not yet necessarily produce a picture of the hand only, as some pixels belonging to the background may pass through the filter whereas parts of the hand that are either shadowed or reflect light are excluded. Hence we need to apply several processing steps; first to extract the hand and then to identify the gesture that the user is currently making.

4.2 Skin-color filtering

Skin color has proven to be a useful and robust cue for face detection, localization, and tracking [Mah08, VSA03, KMB07]. Content filtering, content-aware video compression, and color-balancing applications can also benefit from automatic detection of skin in images. The goal of skin-color detection is to construct a decision rule to discriminate between skin and non-skin pixels. This is usually accomplished by introducing a metric, which measures the distance of the color of a given pixel to a defined value representing skin tone. The specific distance metric employed depends on the skin-color modeling method. An obvious advantage of such methods is the simplicity of the skin-detection rules that enables the implementation of a very fast classifier [VSA03].

Colorimetry, computer graphics, and video-signal transmission standards have given birth to many *color spaces* with different properties. A wide variety of them has been applied to the problem of skin-color modeling. The red-blue-green (RGB) is a color space that originated from cathode-ray tube display applications, where it was convenient to describe each color as a combination of three colored rays: red, green, and blue. This remains one of the most widely-used color spaces for processing and storing of digital image data.

$Y C_B C_R$ is a family of color spaces used as a part of the color-image pipeline in video and digital photography systems. Y is the *luma component*, sometimes called luminance, that represents the brightness in an image. C_B and C_R are the blue-difference and red-difference chroma components; chroma is the signal used in video systems to convey the color information of the picture

In contrast to RGB, the $Y C_B C_R$ color space is luma-independent, resulting in a better performance. $Y C_B C_R$ is not an absolute color space; rather, it is a way of encoding RGB information. The actual color displayed depends on the actual RGB primaries used to display the signal.

The hand-gesture detection algorithm uses skin-color detection. The skin-color filter used in our work can also be used for face detection, localization, and tracking of persons in videos.

Denote by I be the entire input image, and by I_Y , I_{C_B} and I_{C_R} the luma, blue, and red components of the image, respectively. We denote the image height in pixels by h and the image width in pixels by w . The pixel in position (i, j) is denoted by $p_{i,j}$ and its three components by $p_{i,j}^Y$, $p_{i,j}^{C_B}$, and $p_{i,j}^{C_R}$. For all components $C \in \{Y, C_B, C_R\}$, we assume that $p_{i,j}^C \in [0, 255]$, corresponding to eight bits per color channel, yielding 24 bits per pixel. This gives image size of $h \times w \times 24$ bits.

We use a pixel-based skin detection method [KPS03] that classifies each pixel as skin or non-skin individually. More complex methods that take decisions based not only on a pixel $p_{i,j}$, but also on its direct neighborhood $\{p_{i-1,j}, p_{i+1,j}, p_{i,j-1}, p_{i,j+1}\}$ (and possibly also the diagonal neighborhood $p_{i-1,j-1}, p_{i+1,j-1}, p_{i+1,j+1}, p_{i-1,j+1}$) can be formulated, but are computationally heavier. Our aim is to operate the system in real time, for which we seek the simplest and fastest possible method for each step.

A pixel $p_{i,j}$ in I is classified —heuristically, based on existing literature— as skin if all of the following conditions simultaneously apply:

1. The luma component exceeds its corresponding threshold value:

$$p_{i,j}^Y > 80. \quad (1)$$

2. The blue and red components are within their corresponding ranges:

$$\begin{aligned} 85 &< p_{i,j}^{C_B} < 135, \\ 135 &< p_{i,j}^{C_R} < 180. \end{aligned} \quad (2)$$

We write $S(p_{i,j}) = \top$ if the pixel $p_{i,j}$ passes the filter, and $S(p_{i,j}) = \perp$ if it does not fulfill the above conditions.

We then create a new binary image B of the same dimension $w \times h$ (cf. Figure 3 for an example) where the color of the pixel $b_{i,j}$ is either white (denoted by 1) if the position corresponds to skin or black (denoted by 0) if the position did not pass the skin filter:

$$b_{i,j} = \begin{cases} 1, & \text{if } S(p_{i,j}) = \top, \\ 0, & \text{if } S(p_{i,j}) = \perp. \end{cases} \quad (3)$$

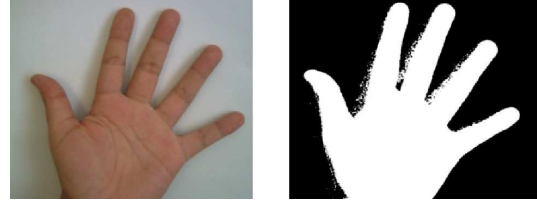


Figure 3: On the left, an original image I . On the right, the resulting binary image B after applying the skin-color filter defined by Equations 1-2.

4.3 Edge detection

Using the binary image corresponding to the presumed skin pixels, we need to determine which of these form the hand, meaning that we need to identify the edge of the hand shape in the image. *Edge detection* is an essential tool in image processing and computer vision, particularly in the areas of feature detection and feature extraction. An *edge* is defined as the boundary between an object and the background, although it may also indicate the boundary between overlapping objects.

The process of edge detection is generally based on identifying those pixels at which the image brightness has discontinuities. When the edges in an image are accurately identified, the objects in it can be located, allowing the computation of basic properties of each object, such as the area, perimeter, and shape [Par96].

There are two main methods used for edge detection; namely the *template matching* and the *differential gradient* methods. In both of these methods, the goal is to identify locations in which the magnitude of the intensity gradient (that is, the change that occurs in the intensity of pixel color when moving across adjacent pixels) is above a threshold, as to indicate in a reliable fashion the edge of an object. The principal difference between the two methods is the way in which they perform local estimation of the intensity gradient g , although both techniques employ convolution masks.

The template matching operates by taking the maximum over a set of component masks (such as the Roberts, Sobel, and Prewitt operators) that represent possible edge configurations. This yields an approximation for g at the pixel in which the templates are centered. The differential gradient method instead computes the pixel magnitudes vectorially with a nonlinear transformation. After computing g for each pixel —with either of these methods— thresholding is carried out to obtain a set of *contour points* (that is, those that were classified as being part of an edge). The orientation of the edges can be deduced from the direction of the highest gradient (the edge being perpendicular to it at that pixel).

At this point, we have the set of contour pixels and need to determine the connected components of the contour, meaning that we must compute the connected

sets of edge points. To create a connected set we select one contour pixel as a seed and recursively add to the set pixels that are also contour pixels and are adjacent to at least one pixel in the set, until there are no more adjacent contour pixels. If there are contour pixels left, then we select another contour pixel as a seed to create a new connected set; we repeat iteratively until all the contour pixels are in a connected component.

In our case, we assume the hand to be in the image foreground, making it likely that the largest connected contour component will correspond to the hand, whereas any smaller components of the contour set, if present, correspond to some objects on the background.

We denote the set of contour pixels of the largest connected component by E . We construct a new binary image O by copying B and then setting to zero (black) all those pixels that correspond to the smaller connected components of the contour and their insides, leaving only E and the pixels inside it at one (white). This can be done by a standard bucket-fill algorithm.

4.4 Convex hull and convexity defects

At this point, we have identified the edge of the hand in the image. We now proceed to determining which hand gesture is being made in the image. The way in which this is done depends on the type of hand gestures supported by the system—no single design is adequate for all possible hand positions—. The gestures that we wish to detect are shown in Figure 4.



Figure 4: The gestures used in our proposed system that correspond to the numbers from zero to five. Note that the separation of the individual fingers is relevant to the detection of these gestures.

As our gestures correspond to distinct numbers of fingers elevated, our detection method is based on counting the elevated fingers in the image. It will not be relevant which finger is elevated, only the number of fingers (cf. [CB03, ÇMÖ06]). This gives us the advantage of the system not being sensitive to which hand is being used, left or right. Additionally we gain not having to control the position of the hand: we can look at the palm or the back and have the person hold his or her arm at nearly any angle with respect to the camera. All we require is that either the palm or the back of the hand faces the camera and that the fingers are separated. This second requirement can be relaxed in future work; we discuss later in this paper how we expect to achieve this.

We identify the peaks of the fingers in the image by computing the *convex hull* of the hand edge. The convex hull is a descriptor of shape, is the smallest convex set that contains the edge; intuitively explained—in two dimensions—as the form taken by a rubber band when placed around the object in question; an example is given in Figure 5). It is used in computer vision to simplify complex shapes, particularly to provide a rapid indication of the extent of an object.

We now copy the binary image O to a new image C . We will then iteratively seek and eliminate *concave* regions. Intuitively, this can be done by examining the values of the pixels in an arbitrary straight segment with both endpoints residing in white pixels. If any of the pixels along the segment are black, they are colored white, together with any black pixels beneath the segment. This repeated “filling” will continue until no more segments with white end points and intermediate black pixels exist. An algorithm for achieving this is given in the text book of Davies [Dav04].

The resulting white zone in C is now *convex* and the edge of that zone—all those white pixels that have at least one black neighbor—form the convex hull of the hand-shape in E . We denote this edge by H .

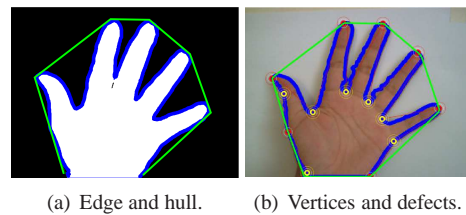


Figure 5: On the left, the background (in black), the hand-shape region O (in white), the hand edge E (in blue), and the convex hull H (in green). On the right, we add the vertices of the convex hull (in red) and the convexity defects (in yellow).

We now proceed to comparing H to E to detect the *defects*, points in which the two differ greatly. First, from H , we compute the *vertices* of the convex hull, that is, the points in which it changes direction. Then, we examine the segments of E between pairs of consecutive vertices of H and find that pixel in each segment that maximizes the distance from H . This maximal distance d_i is called the *depth* of the defect i . The points themselves are called *convexity defects*. Figure 5 shows an example.

From the defect depths, useful characteristics of the hand shape can be derived, such as the depth average μ_d . We use the defect depths, together with the depth average and the total hand length, to count the number of elevated fingers. An above-average depth indicates a gap between fingers, whereas a clearly below-average depth is not a finger separation. Using the number of defects we can estimate the number of elevated fingers

on the hand: an open hand showing five fingers has four convexity defects, whereas a hand showing four fingers has three convexity defects, and so forth.

5 Material and methods

We used OpenCV² under Python³ to implement a prototype of the proposed hand-gesture detection system. As we wanted the system to be able to run on modest hardware, we performed all our experiments on a netbook with a 1.6 GHz processor and 1 GB of RAM memory, using a webcam with a resolution of 640×480 pixels. The prototype operates in real time and indicates on the screen the detected gesture; Figure 6 shows a screen capture.

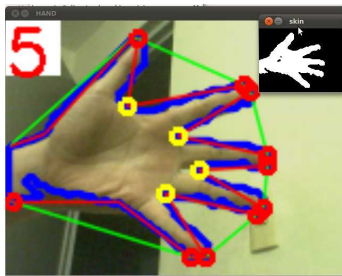


Figure 6: A screen capture of the implemented prototype for the hand-gesture detection tool.

5.1 Experimental setup

We carried out experiments with users to evaluate the functionality of the proposed gesture-detection algorithm. We requested the users to carry out a series of gestures in front of the webcam and measured whether the detection was successful. An observer recorded whether the output produced by the algorithm corresponded to the actual gesture being made. The lighting, camera position, and image background were controlled, as illustrated in Figure 7. We hope to relax these requirements in future work, as the proposed method is designed to accommodate a less restricted use setting.

The user was shown a gesture sequence —on a computer screen (see Figure 8 for an example)—. Each gesture sequence contains a randomly permuted sequence of hand gestures to perform. The sequence was available on the screen while the user performed the gestures one at a time. We instructed the users to take a three-second pause between gestures. Each sequence was performed once with the right hand and then again with the left hand. When the user finished to perform the last gesture in the sequence, a new random sequence was shown. Each user carried out five different sequences.

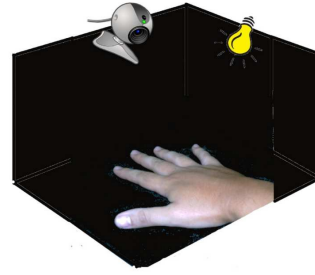


Figure 7: The experimental setting: our arrangement for controlled background, fixed camera position, and constant illumination.



Figure 8: An example of a random gesture sequence assigned to a user.

6 Results of user experiments

We evaluated the prototype with ten users; each performed five sequences of gestures with both hands (each sequence was composed of six gestures from zero to five, in random order). Therefore, each user performed 60 gestures, giving us a total of 600 gesture-detection attempts. Table 6 shows the percent of gestures correctly detected, grouped by the gesture made and the hand used.

Hand used	Gesture detected						Total
	0	1	2	3	4	5	
Right hand	100%	72%	96%	94%	98%	100%	93.33%
Left hand	100%	76%	94%	96%	98%	94%	93.00%
Total	100%	74%	95%	95%	98%	97%	93.17%

Table 1: Percentage of correctly identified gestures.

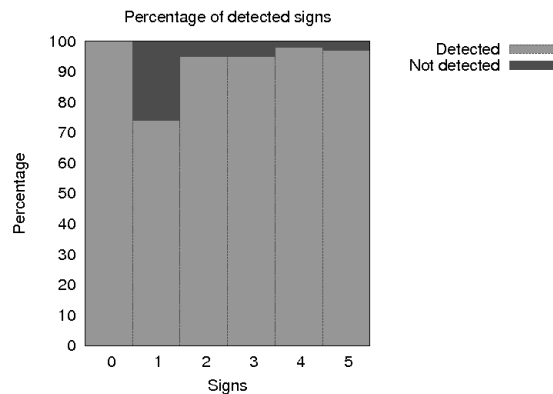


Figure 9: Correctly detected gestures.

² <http://opencv.willowgarage.com/>

³ <http://www.python.org/>

In total, 93.1% of the gestures were correctly detected, improving the results for a previous work [RSS12]; the gestures for numbers three, four, and five have the highest accuracy and present low variation between hands. The gestures for number one, however, has the lowest detection percentage. Also, gestures for zero, one, and two show variability according to the hand used. The gesture-detection algorithm works correctly a majority of the time, under the conditions used in our experiments. User observation helped us notice that the primary cause for incorrect gesture detection was the particular form in which each user performs the gesture: sometimes, for example, the fingers were very close to each other. Some examples are shown in Figure 10. We discuss a possible work-around to this problem as part of future work in the next section.

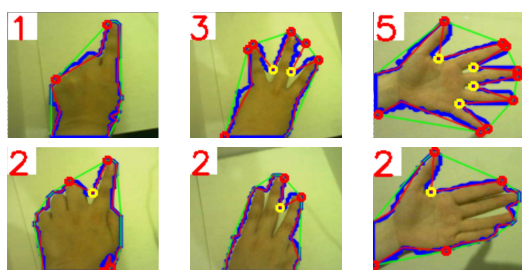


Figure 10: Some examples of correct and incorrect detection from the user experiments. Above, a correctly detected gesture, and below, an incorrect detection of that same gesture. The gestures requested were, from left to right, one, three, and five.

7 Conclusions and future work

We have presented a method for detecting hand gestures based on computer-vision techniques, together with an implementation that works in real time on an ordinary webcam. The method combines skin-color filtering, edge detection, convex-hull computation, and a rule-based reasoning with the depths of the convexity defects. We had reported as well user experiments on the detection accuracy of the developed prototype, detecting correctly nine in ten hand gestures made on either hand, in a controlled environment.

As future work, we plan to add in the gesture detection phase an estimate of the width of each finger. This allows us to determine whether a single finger is elevated at that position or whether multiple fingers are elevated but held together. The finger width can be calibrated for each person by measuring the width of the hand base itself and assuming that anything that has the width between one sixth and one fourth of the base width is a single finger. The number of fingers in a wider block can be estimated as the width of the block (computable from the points used for finger counting at

present) divided by one fifth of the base width, rounded down to the preceding integer value.

Another aspect that needs to be addressed in future work is the sensibility of the system to lighting conditions, as this affects the skin-color filtering, particularly with reflections and shadows. We expect these additions to improve the accuracy of the detection system, as well as ease the cognitive burden of the end user as it will no longer be necessary to keep the fingers separate—something that one easily forgets—.

8 References

REFERENCES

- [AL06] Antonis Argyros and Manolis Lourakis. Vision-based interpretation of hand gestures for remote control of a computer mouse. In Thomas Huang, Nicu Sebe, Michael Lew, Vladimir Pavlovic, Mathias Kölsch, Aphrodite Galata, and Branislav Kisanin, editors, *Computer Vision in Human-Computer Interaction*, volume 3979 of *Lecture Notes in Computer Science*, pages 40–51. Springer, Berlin / Heidelberg, Germany, 2006.
- [AM06a] Farooq Ahmad and Petr Musilek. A keystroke and pointer control input interface for wearable computers. In *IEEE International Conference on Pervasive Computing and Communications*, pages 2–11, Los Alamitos, CA, USA, 2006. IEEE Computer Society.
- [AM06b] Farooq Ahmad and Petr Musilek. Ubihand: a wearable input device for 3D interaction. In *ACM International Conference and Exhibition on Computer Graphics and Interactive Techniques*, page 159, New York, NY, USA, 2006. ACM.
- [CB03] Stephen C. Crampton and Margrit Betke. Counting fingers in real time: A webcam-based human-computer interface game applications. In *Proceedings of the Conference on Universal Access in Human-Computer Interaction*, pages 1357–1361, Crete, Greece, June 2003. HCI International.
- [ÇMÖ06] Müdjat Çetin, Asanterabi Kighoma Malima, and Erol Özgür. A fast algorithm for vision-based hand gesture recognition for robot control. In *Proceedings of the IEEE Conference on Signal Processing and Communications Applications*, pages 1–4, NJ, USA, 2006. IEEE.
- [Dav04] E. Roy Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
- [EBN⁺07] Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, and Xander Twombly.

- Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108:52–73, 2007.
- [HNW⁺09] Yuki Hirobe, Takehiro Niikura, Yoshihiro Watanabe, Takashi Komuro, and Masatoshi Ishikawa. Vision-based input interface for mobile devices with high-speed fingertip tracking. In *22nd ACM Symposium on User Interface Software and Technology*, pages 7–8, New York, NY, USA, 2009. ACM.
- [KMB07] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3):1106–1122, 2007.
- [Kol10] Kolsch. Handvu. www.movesinstitute.org/textasciitildekolsch/HandVu/HandVu.html, 2010.
- [KPS03] J. Kovac, P. Peer, and F. Solina. Human skin colour clustering for face detection. In *International conference on Computer as a Tool*, volume 2, pages 144–147, NJ, USA, 2003. IEEE.
- [LBT02] S. Lenman, L. Bretzner, and B. Thuresson. Computer vision based hand gesture interfaces for human-computer interaction. Technical report, CID, Centre for User Oriented IT Design. Royal Institute of Technology Sweden, Stockholm, Sweden, June 2002.
- [Mah08] Tarek M. Mahmoud. A new fast skin color detection technique. *World Academy of Science, Engineering and Technology*, 43:501–505, 2008.
- [MHP⁺01] J. MacLean, R. Herpers, C. Pantofaru, L. Wood, K. Derpanis, D. Topalovic, and J. Tsotsos. Fast hand gesture recognition for real-time teleconferencing applications. In *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pages 133–140, Washington, DC, USA, 2001. IEEE Computer Society.
- [MMC09] Pranav Mistry, Pattie Maes, and Liyan Chang. WUW - wear ur world: a wearable gestural interface. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4111–4116, New York, NY, USA, 2009. ACM.
- [Par96] J. R. Parker. *Algorithms for Image Processing and Computer Vision*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 1996.
- [RPJ10] S.M.M. Roomi, R.J. Priya, and H. Jayalakshmi. Hand gesture recognition for human-computer interaction. *Journal of Computer Science*, 6(9):1002–1007, 2010.
- [RSS12] David J. Rios Soria and Satu E. Schaeffer. A tool for hand-sign recognition. In *4th Mexican Conference on Pattern Recognition*, volume 7329 of *Lecture Notes in Computer Science*, pages 137–146. Springer, Berlin / Heidelberg, 2012.
- [SP09] E. Stergiopoulou and N. Papamarkos. Hand gesture recognition using a neural network shape fitting technique. *Engineering Applications of Artificial Intelligence*, 22(8):1141–1158, 2009.
- [SPHB08] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a Wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14, New York, NY, USA, 2008. ACM.
- [VSA03] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *Proceedings of international conference on computer graphics and vision*, pages 85–92, Moscow, Russia, 2003. Moscow State University.
- [WKSE11] Juan Pablo Wachs, Mathias Kölsch, Helman Stern, and Yael Edan. Vision-based hand-gesture applications. *Communications ACM*, 54:60–71, feb 2011.
- [WMW09] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1083–1092, New York, NY, USA, 2009. ACM.
- [WP09] Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 28:63:1–63:8, jul 2009.
- [WSE⁺06] Juan Wachs, Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith, and Jon Handler. A real-time hand gesture interface for medical visualization applications. In Ashutosh Tiwari, Rajkumar Roy, Joshua Knowles, Erel Avineri, and Keshav Dahal, editors, *Applications of Soft Computing*, volume 36 of *Advances in Soft Computing*, pages 153–162. Springer, Berlin / Heidelberg, 2006.
- [ZM11] Morteza Zahedi and Ali Reza Manashty. Robust sign language recognition system using ToF depth cameras. *Information Technology Journal*, 1(3):50–56, 2011.

Neural-Based Segmentation Technique for Arabic Handwriting Scripts

Husam A. Al Hamad

College of Computer

Qassim University

Saudi Arabia

hushamad@yahoo.com, hhamad@qu.edu.sa

ABSTRACT

In some algorithms, segmentation of the word image considers the first step of the recognition processes; the main aim of this paper is proposed new fusion equations for improving the segmentation of word image. The technique that has used is divided into two phases; at the beginning, applying the Arabic Heuristic Segmenter (AHS), AHS uses the shape features of the word image, it employs three features, remove the punctuation marks (dots), ligature detection, and finally average character width, the goal of this technique is placed the Prospective Segmentation Points (PSP) in the whole parts of the word image. As a result, the second phase apply the neural-based segmentation technique, the goal of neural technique is check and examine all PSPs in the word image in order to report which one is valid or invalid, this will increase the accuracy of the segmentation; to do that, the network obtains a fused value from three neural confidences values: 1) Segmentation Point Validation (SPV), 2) Right Character Validation (RCV), and 3) Central Character Validation (CCV) which will assess each PSP separately. The input vectors of the neural network are calculated based on Direction Feature (DF), DF considers much more suitable for Arabic Scripts. AHS and neural-based segmentation techniques have been implemented and tested by local benchmark database.

Keywords:

Arabic handwriting recognition, neural networks, Arabic heuristic segmenter.

1. INTRODUCTION

The concept of handwriting recognition can be divided according to [Pla01a] into two main areas, these areas are on-line and off-line. An off-line Arabic handwriting segmentation and recognition is one of the most challenging researches because there are different variations in handwriting [Naw01a], it is an approach that interprets characters, words and scripts that have been written at common surface (i.e. paper). On the other hand, on-line handwriting recognition refers to automatically recognizing the handwritten characters using real-time information such as pressure and the order of strokes made by a writer usually employing a stylus and pressure sensitive tablet [Cas01a, Lor01a].

The segmentation [Bal01a, Man01a] of Arabic handwritten characters have been an area of great interest in the past few years [Blu00a]. One typical

approach in the literature is “over-segmentation” which is known as dissecting the word image based on shape features of the image into a sufficient number of components; so that no merged characters remain [Yan01a, Xia01a]. One of the major problems following over-segmentation is correctly discard the invalid segmentation points and remained the valid points, to determine the valid segmentations, many of researches are studied by merging segments of the image and invoking a classifier to score the combinations, the most techniques employ the optimization algorithms that making use some sort of lexicon-driven and dynamic programming technique [Blu00a]. The best way to evaluate the over-segmentation is use the neural networks [Fan01a], the most common family of neural networks for pattern classification recognition is Feed-Forward Back-Propagation network (FFBP) which is very simple and effective to implement, it has been applied successfully to different applications domains, such as pattern recognition, controlling, prediction, system identification, etc. [Bil01a], the weight inputs transmits to the neurons in the first layer and the neurons transmits their outputs to the neurons in to the next layer, etc., the network not contain any cycles or loop as an advantage [Abd01a].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2. LITERATURE REVIEW

Earlier art showed segmentation of both machine-print and handwriting. In 1980, Nouh *et al.* suggested a standard Arabic character set to facilitate computer processing [Nou01a]. Sami El-Dabi *et al.* used segmented characters based on invariant moments only after they were recognized. Recognition was attempted on regions of increasing width until a match was found [Dab01a]. Yamin and Aoki presented a two-step segmentation system which used vertical projection onto a horizontal line followed by feature extraction and measurements of character width [Ymi01a]. Al-Badr and Haralick presented a holistic recognition system based on shape primitives that were detected with mathematical morphology operations [Bad01a]. Hamami and Berkani developed a structural approach to handle many fonts, and it included rules to prevent over-segmentation [Ham00e]. Al-Qahtani and Khorsheed presented a system based on the portable Hidden Markov Model Toolkit [Qah01a]. Srihari and Ball, applied heuristic techniques for image processing representation of the binary image counter and removal of noise and dots [Sri01a]. Hamad and Zitar [Ham00c] applied new fusion equations in order to enhance the segmentations processes. Hamad [Ham00d] developed a technique that aim to assign the prospective segmentation points which is obtained based on the shape features of word image. On the other hand, many researches are using the feed-forward back-propagation neural network, the origin of this type is used by Rumelhart [Rum01a] in 1986, the application area network of back-propagation algorithm are gained recognition and utilized multiple layers of weight-sum units of the type $f = g(w \cdot x + b)$. Training was done by a form of stochastic gradient descent.

3. PROBLEMS OF ARABIC SCRIPTS

Many researches have been published in the area of

handwritten Arabic scripts recognition [Ham00a, Ham00b], so far, the researches haven't been reached to good result because it is considerably harder due to a number of reasons: 1) Arabic is written cursively, i.e., more than one character can be written connected to each other. 2) Arabic uses many types of external objects, such as dots, "Hamza", "Madda", and diacritic objects. These make the task of line separation and segmentation scripts more difficult. 3) Arabic characters can have more than one shape according to their position: initial, middle, final, or stand alone. 4) Characters that do not touch each other but occupy a shared horizontal space that increases the difficulty of segmentation, 5) Arabic uses many ligatures, especially in handwritten text, this makes the segmentation of Arabic scripts even more difficult [Ham00c].

4. SEGMENTATION TECHNIQUE

Arabic Heuristic Segmenter (AHS) or over-segmentation technique aims to assign correct PSP points in the word image [Nic01a]. Following this, a neural confidence-based module has been used to validate these points by obtaining a fused value from three neural confidence values based on Segment Point (SP), Right Character (RC), and Central Character (CC) [Che00a]. Segmentation technique has two advantages; first, reducing the number of missed or bad points, and second, increasing the accuracy of the recognition rate. Since number of segmentation points is optimized by using this technique, the overall accuracy will increase and processing time will reduce [Che00b]. Missed points occur when no segmentation point is determined between two successive characters; besides, bad points refer to the points that could not be used to extract the characters precisely. AHS which was proposed by Hamad [Ham01c, Ham00d] removed the punctuation marks (dots) that hinder identify the correct segment points, this technique helps to detect

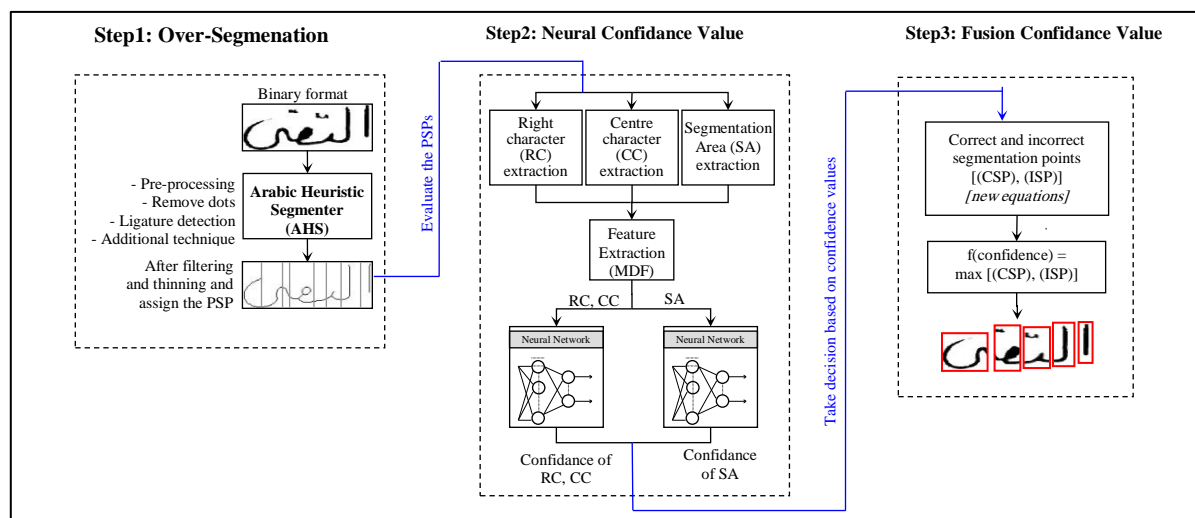


Figure 1. Overview of the segmentation technique

the ligatures that connect between two successive characters to obtain the correct segmentation points. Additional techniques such as average character width are applied as well to enhance the results. One of the major problems following over-segmentation is correctly discard the invalid segmentation points and retained the valid one by using neural network, the input vector of the network is extracted based on Modified Direction Feature (MDF) [Blu00b]. Figure 1 illustrates an overview of the entire neural-based segmentation technique.

4.1. Over-Segmentation

Over-Segmentation or AHS employs three techniques: 1) Pre-processing, filtering the word image, and removing the punctuation marks or any redundant components. 2) Ligature detection, a ligature is a small point (stroke) that is used to connect between two characters; the aim of ligature detection is locate these strokes within the “middle region” of handwritten words. 3) Calculates the average character width, the technique aims to add any missing segment points and remove the bad one, an addition technique is detect the close and open holes which is aims to remove any bad points across these holes that considered complete characters. The results of these techniques are word image contains a sufficient number of PSPs; these points will be evaluated by the neural networks later.

4.2. Modified Direction Features (MDF)

Arabic handwritten has a special characteristics such as rotations, curves, and circuits shapes; so, the suitable features input in the vectors of neural network is direction features, MDF extracts the direction information (feature) from structure of the character contours that determined in each character image, the technique categorizes into four parts: 1) Vertical lines, 2) Horizontal lines, 3) Right diagonal, and finally 4) Left diagonal. This principle is extended so that integrates the direction feature with the technique for calculate the transition features between background pixels (white pixels) and foreground pixels (black pixels). In MDF, Location of Transitions (LTs), and Direction Transition (DT) are calculated at a particular location, therefore, for each transition, a pair of values such as [LT, DT] are stored; this work demonstrated the superiority of MDF for describing the Arabic patterns according to their contour or boundary. More details have been described in [Blu00a, Ham00c].

4.3. Neural-based Validation

As a result of above and after completing the over-segmentation, the post-processing is employed to exclude the bad segment point and remain the correct. The classifier chosen for the validation is a feed-forward neural network trained with the back-propagation algorithm. For experimental purposes,

the architectures were modified varying the number of inputs, outputs and hidden units. Three vectors are extracted from the word image to validate each PSP and determine whether correct or not, where the classifier will calculate and output the confidence value for each point, the values represent each of the segmentation area (SA), right character (RC), and center character (CC) and validate all of them based on maximum of confidence value. Therefore, it is possible to validate prospective segmentation points, rather than giving a binary result (valid or invalid) decision whether a segmentation point should be set in a particular region, confidence values are assigned to each segmentation points that are located through feature detection. The confidence value of any segment area should be in the range of 0 and 1.

4.4. Fusion Confidence Values

Fusion confidence value is a set of equations take the final decision (valid or invalid), where are calculated on the basis of the output confidence value of the neural network. New fusion equations are proposed, the extracted areas of these equations are analyzed and described as: Rule 1: Following RC extraction and neural verification, the area is analyzed into two options: 1) If the area is identified by the neural expert as one of 62 possible characters, then the segmentation point is more likely to be a correct segmentation point. 2) If the area is identified as a non-character (rejected), then the segmentation point is more likely to be an incorrect segmentation point. Rule 2: Following CC extraction and neural verification, the area is analyzed into two options: 1) If the area is identified by the neural expert as one of 62 possible characters, then the segmentation point is more likely to be an incorrect segmentation point. 2) If the area is identified as a non-character then the segmentation point is more likely to be a correct segmentation point. Rule 3: Following SA extraction, the area is analyzed into two options: 1) If the neural expert provides a confidence ≥ 0.5 , then the segmentation point is more likely to be a correct segmentation point. 2) If the neural expert provides a confidence < 0.5 , then the segmentation point is more likely to be an incorrect segmentation point

Two possibilities for each fusion are applied, first, calculate Correct Segmentation Point (CSP) where Segmentation Point Validation (SPV) ≥ 0.5 as shown in equation 1; second, calculate Incorrect Segmentation Point (ISP) where $SPV < 0.5$ as shown in equation 2; finally, calculate outcome of the fusion decision based on maximum value between the CSP and ISP as shown in equation 3. If the CSP confidence is greater, then the SP will be set as being correct. Conversely, if the ISP confidence prevails as being larger, the SP will be discarded and no longer used in further processing.

1) Correct Segmentation Point (CSP):

if $f_{SPV_ver}(ft1) \geq 0.5$ AND $f_{RCC_ver}(ft2)$ is a high character confidence AND $f_{CCC_ver}(ft3)$ is a high non-character confidence, then:

$$f_{CSP}(ft1, ft2, ft3) = f_{SPV_ver}(ft1) + f_{RCC_ver}(ft2) + (1 - f_{CCC_ver}(ft3)) \quad (1)$$

2) Incorrect Segmentation Point (ISP):

if $f_{SPV_ver}(ft1) < 0.5$ AND $f_{RCC_ver}(ft2)$ is a high non-character confidence AND $f_{CCC_ver}(ft3)$ is a high character confidence, then

$$f_{ISP}(ft1, ft2, ft3) = f_{SPV_ver}(ft1) + (1 - f_{RCC_ver}(ft2)) + f_{CCC_ver}(ft3) \quad (2)$$

3) Finally, the outcome of the fusion is decided by the following equation:

$$f(\text{confidence}) = \max [(CSP), (ISP)] \quad (3)$$

Where, $f_{SPV_ver}(\text{features})$ is confidence value of Segmentation Point Validation, $f_{RCC_ver}(\text{features})$ is a confidence value for right character, and $f_{CCC_ver}(\text{features})$ is confidence value for center character (reject neuron output).

Original Word	Over-segmentation	Segmentation
بدرجات		
دكتور		
مدرسة		
تفسير		
افضل		
تقديرية		
مساعدة		
وفوائده		
مسيرة		
الثقافية		

(a) successful segmentation

Original Word	Over-segmentation	Segmentation
الطلاب		
اساس		
درشد		
أحدث		
لصقل		

باسم		
فايكر سوفت		
وزارات		
العديد		
وقال		

(b) unsuccessful segmentation

Figure 2. Segmented sample of word images

5. EXPERIMENTAL RESULTS

The experiments here used the neural confidence-based module for validating the PSPs which are obtained from AHS (over-segmentation). Segmentation performance is measured based on three types of segmentation errors: “over-segmentation”, “missed” and “bad” metrics. Over-segmentation refers to a character that has been divided into more than three components. A “missed” error occurs when no segmentation point is found between two successive characters. The “bad” error refers to a segmentation point that could not be used to extract a character precisely.

5.1. Handwriting Database

The training and testing patterns samples were obtained and extracted from twenty different persons, all words are selected randomly. They were asked to write down two paragraphs contains all status of Arabic characters. These paragraphs scanned at 200 pixels per inch. The size of training set was 620 characters (10 writers x 62 characters), and size of testing set was 425 words, more details about the database see www.acdar.org.

5.2. AHS Segmentation Performance

The total numbers of segmentation points in the 425 testing word samples are 3080. Table 1 shows the segmentation performance of the AHS technique, see [Ham00d] for more details about this results.

Result		Segmentation Error Rates			
		Over Seg.	Missed	Bad	Bad/overlap
Totals		29	18	552	26
%		0.94%	0.58%	17.92%	0.84%
With overlap	Total	599			
	%	19.45%			
Without overlap	Total	573			
	%	18.60%			

Table 1. AHS segmentation error

5.3. Neural-based Performance

Results of the neural-based segmentation technique were calculated based on the number of correct and incorrect identified of segment point in word samples. Neural network verifies whether

segmentation points are valid or invalid based on neural confidence-based module. If the network output a height confidence value this indicated that a point is a valid segmentation point; a low confidence value indicated that a point should be ignored, Table 2 illustrates the overall results of the technique.

Result		Correctly Identified		Incorrectly Identified		
		Valid	Invalid	Valid	Invalid	Invalid overlap
Totals		2011	729	192	148	40
%		65.29%	23.67%	6.23%	4.81%	1.30%
With overlap	Total	2740		340		
	%	88.96%		11.04%		
Without overlap	Total	2780		300		
	%	90.26%		9.74%		

Table 2. Results of neural-based segmentation technique

The above results describe the recognition rate for the neural networks. To enhance these rates, the number in the testing set must be increased at least two or three-fold, that will help improving overall segmentation accuracy, Figure 3 illustrates the characters recognition rates of the neural-based segmentation technique.

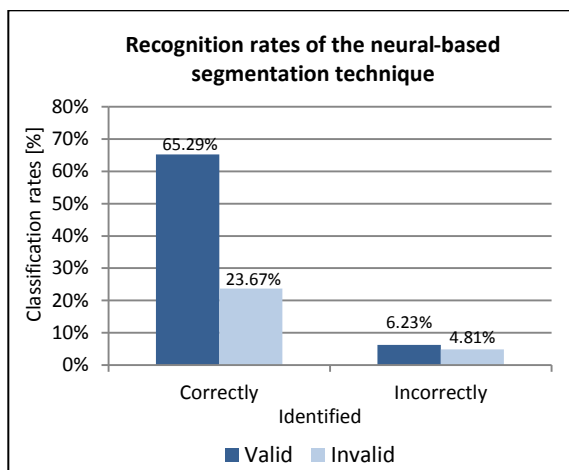


Figure 3. Recognition rates for all different neural networks.

Table 3 shows the summary of the literature results and the comparisons with the paper's results.

Reference	Accuracy	Language / Databases
Blumenstein, Myer [Blu00c]	75.28%	<ul style="list-style-type: none"> Cursive English handwriting CEDAR database
Hamid, Alaa [Ham00a]	69.72%	<ul style="list-style-type: none"> Arabic handwriting Local database: 360 addresses, 4000 images
Cheng, Chun Ki [Che01a]	85.74%	<ul style="list-style-type: none"> Cursive English handwriting CEDAR database: test 1031 from 1718 SP
Khateeb, Jawad [Kha01a]	85.00%	<ul style="list-style-type: none"> Arabic handwriting Local database: 200 images, sub-words SP

Hamad, Husam Al [Ham00d]	82.98%	<ul style="list-style-type: none"> Arabic handwriting Local database: 500 images
This paper	88.96%	<ul style="list-style-type: none"> Arabic handwriting Local database: 425 images

Table 3. Compare the results with the literature

6. CONCLUSIONS

This paper investigates collection of techniques aims to segmenting the Arabic handwritten scripts, new fusion equations, and heuristic technique are developed, the technique splits the word image into a sufficient number of components, in order to separate the word image into its characters, the technique called "over-segmentation" or Arabic heuristic segmenter (AHS). Modified Direction Features (MDF) is also employed which is considered a promised technique for Arabic scripts, MDF extracts the input vector feature of the neural network, the AHS provides better inputs to the subsequent neural validation process. Promised results were obtained in this study may increase the performance of a segmentation-based handwriting recognition systems. In the future, a larger size of training set will investigated in order to improve the results of the classifiers as well as reduce the errors.

7. REFERENCES

- [Abd01a] Abdalla, O.A., and Zakaria, M.N., and Sulaiman, S., and Ahmad, and W.F.W.: A comparison of feed-forward back-propagation and radial basis artificial neural networks: A Monte Carlo study, *Information Technology (ITSim)*, vol. 2, pp.994-998, 2010.
- [Bad01a] Al-Badr, B., Haralick, R.: A Segmentation-Free Approach to Text Recognition with Application to Arabic Text, *International Journal on Document Analysis and Recognition*, vol. 1, pp. 147-166, 1998.
- [Bal01a] Ball, G., Srihari, S., Srinivasan, H.: Segmentation-Based and Segmentation-Free Methods for Spotting Handwritten Arabic Words, In: *IWFHR*, 2006.
- [Bil01a] Bilski, J.: The Ud Rls Algorithm for Training Feedforward Neural Networks, *Int. 1. Appl. Math. Comput. Sci.*, pp. 115-123, 2005.
- [Blu00a] Blumenstein M., Liu X.Y., Verma, B.: An investigation of the modified direction feature for cursive character recognition. *Pattern Recognition*. vol. 40(2), pp. 376-388, 2007.
- [Blu00b] Blumenstein, M., Liu, X.Y., Verma, B.: A Modified Direction Feature for Cursive Character Recognition. *International Joint Conference on Neural Networks*. Budapest, Hungary, pp. 2983-2987, 2004.
- [Blu00c] Blumenstein, Myer. *Intelligent Techniques for Handwriting Recognition*, School of

- Information Technology, PhD Dissertation, Griffith University-Gold Coast Campus, Australia, 2000.
- [Cas01a] Casey, R., Lecolinet, E.: A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Analysis and Mach. vol. 18*, pp. 690-706, 1996.
- [Che00a] Cheng, C.K., Blumenstein, M.: The Neural-based Segmentation of Cursive Words using Enhanced Heuristics. In: Eighth International Conference on Document Analysis and Recognition, pp. 650-654, 2005.
- [Che00b] Cheng, C.K., Liu, X.Y., Blumenstein, M., Muthukumarasamy, V.: Enhancing Neural Confidence-Based Segmentation for Cursive Handwriting Recognition. In: 5th International Conference on Simulated Evolution and Learning Busan, Korea, SWA-8, CD-ROM Proceedings, 2004.
- [Dab01a] El-Dabi, S., Ramsis, R., Kamel, A.: Arabic Character Recognition System: A Statistical Approach for Recognizing Cursive Typewritten Text, *Pattern Recognition*, vol. 23, pp. 485-495, 1990.
- [Fan01a] Fan, X., Verma, B.: Segmentation vs. Non-Segmentation Based Neural Techniques for Cursive Word Recognition. An Experimental Analysis. *International Journal of Computational Intelligence and App.* vol. 2(4), p.p. 377-384, 2002.
- [Ham00a] Hamid, A., Haraty, R.: A Neuro-Heuristic Approach for Segmenting Handwritten Arabic Text, In: ACS/IEEE International Conference on Computer Systems and Applications, p.p. 0110, 2001.
- [Ham00b] Hamid, A., Haraty, R.: Segmenting Handwritten Arabic Text. *ACIS International Journal of Computer and Information Science*, vol. 3 (4), 2002.
- [Ham00c] Hamad, H.A., Zitar, R.: Development of an efficient neural-based segmentation technique for Arabic handwriting recognition. *Pattern Recognition Journal. ELSEVIER*. vol. 43, Issue 8, p.p. 2773-2798, 2010.
- [Ham00d] Hamad, Husam A. Al: Over-segmentation of handwriting Arabic scripts using an efficient heuristic technique, In: Wavelet Analysis and Pattern Recognition (ICWAPR), IEEE, pp.180-185, 2012.
- [Ham00e] Hamami, L., Berkani, D.: Recognition System for Printed Multi-font and Multisize Arabic Characters, the *Arabian Journal for Science and Engineering*, vol. 27, pp. 57-72, 2002.
- [Kha01a] Jawad H AlKhateeb, and Jianmin Jiang, and Jinchang Ren, and Stan S Ipson. Component-based Segmentation of Words from Handwritten Arabic Text, *Proceedings of World Academy of Science, Engineering and Technology*, ISSN, vol. 31, pp. 1307-6884, 2008.
- [Lor01a] Lorigo, L., Govindaraju, V.: Off-line Arabic Handwriting Recognition: A Survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28 (5), p.p. 712-724, 2006.
- [Man01a] Mansour, M., Benkhadda, M.: Optimized segmentation techniques for Arabic handwritten numeral character recognition. In: SITIS, p.p. 96-101, 2005.
- [Naw01a] Nawaz, S.N., Sarfraz, M., Zidouri, A.; Al-Khatib, W.G.: An approach to offline Arabic character recognition using neural networks. *Electronics, Circuits and Systems*, 2003. ICECS 2003. Proceedings of the 2003 10th IEEE International Conference on, vol. 3, p.p. 1328-1331, 2003.
- [Nic01a] Nicchiotti, G., Scagliola, C.: A Simple and Effective Cursive Word Segmentation Method. *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, Amsterdam, pp. 499-504, 2000.
- [Nouh01a] Nouh, A., Sultan, A., and Tolba, R.: An Approach for Arabic Characters Recognition, *J.Eng. Sci., Univ. Riyadh*, vol. 6, pp. 185-191, 1980.
- [Pla01a] Plamondon, R., Srihari, S.N.: On-Line and Off-Line Handwriting Recognition. A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, p.p. 6384, 2000.
- [Qah01a] Al-Qahtani, S., Khorsheed, M., A HTK-Based System to Recognise Arabic Script, in *Proc. 4th IASTED International Conference on Visualization, Imaging, and Image Processing*. Marbella, Spain: ACTA Press, 2004.
- [Rum01a] Rumelhart, David, E., Hinton, Geoffrey, E.; Williams, Ronald J., "Learning representations by back-propagating errors", *Nature*, vol. 323(6088), pp. 533-536, 1986.
- [Sri01a] Srihari, S., Ball, G.: An Assessment of Arabic Handwriting Recognition Technology, in *IWFHR*, CEDAR Technical Report TR-03-07, 2007.
- [Xia01a] Xiao, X., Leedham, G.: Knowledge-based Cursive Script Segmentation. *Pattern Recognition Letters*, vol. 21, pp. 945-954, 2000.
- [Yan01a] Yanikoglu, B., Sandon, P.A.: Segmentation of Off-Line Cursive Handwriting using Linear Programming. *Pattern Recognition*, vol. 31, pp. 1825-1833, 1998.
- [Ymi01a] Ymin, A., Aoki, Y., On the Segmentation of Multi-font Printed Uyghur Scripts, in *Proc. 13th International Conference on Pattern Recognition*, vol. 3, pp. 215-219, 1996.

Polynomiography and various convergence tests

Krzysztof Gdawiec

Institute of Computer Science

University of Silesia

Bedzinska 39

41-200, Sosnowiec, Poland

kgdawiec@ux2.math.us.edu.pl

ABSTRACT

The aim of this paper is to present a modification of the visualization process of finding the roots of a given complex polynomial which is called polynomiography. The name polynomiography was introduced by Kalantari. The polynomiographs are very interesting both from educational and artistic points of view. In this paper we are interested in the artistic values of the polynomiography. The proposed modification is based on the change of the usual convergence test used in the polynomiography, i.e. using the modulus of a difference between two successive elements obtained in an iteration process, with the tests based on distance and non-distance conditions. Presented examples show that using various convergence tests we are able to obtain very interesting and diverse patterns. We believe that the results of this paper can enrich the functionality of the existing polynomiography software.

Keywords

polynomiography, convergence, Basic Family, computer art

1 INTRODUCTION

One of the most elusive goals in computer aided design is artistic design and pattern generation. Pattern generation involves diverse aspects: analysis, creativity, development. A designer have to deal with all of these aspects in order to obtain an interesting pattern which later could be used in jewellery design, carpet design, as a texture etc. Therefore, it is highly motivating and useful to develop new methods of obtaining very diverse and interesting patterns. One place where we can search for this kind of methods is mathematics [Pic01].

Polynomials are one of the mathematical objects which can generate very diverse and beautiful patterns. The patterns from polynomials are often generated through polynomiography. It visualizes the process of finding roots of a complex polynomial using the numerical methods. In this paper we are not interested in the improvement of the numerical methods convergence, but in the artistic aspect of the polynomiography. This aspect includes: creating paintings, carpet design, tapestry design, animations etc. [Kal05b]. So we are interested in obtaining new and interesting patterns basing on the theory of polynomiography.

The paper is organized as follows. In section 2 we introduce the basics of polynomiography. At first we define the Basic Family and give an efficient algorithm for computation of a value for a given element of this family and an algorithm for computation of polynomiograph. The section ends with some examples of polynomiographs. Next, in section 3 we introduce different kinds of convergence test which can be used in the algorithm of polynomiograph computation. In section 4 we show some examples of polynomiographs obtained using the proposed convergence tests. Finally, in section 5 we give concluding remarks and plans for the future work.

2 POLYNOMIOGRAPHY

Polynomiography was introduced by Kalantari about 2000. It is "the art and science of visualization in approximation of the zeros of complex polynomials, via fractal and non-fractal images created using the mathematical convergence properties of iteration functions" [Kal04]. Single image created using the mentioned methods is called polynomiograph. In 2005 Kalantari obtained an U.S. patent on the use of polynomiography in the generation of aesthetic patterns [Kal05a].

In mathematics polynomials are fundamental objects with very diverse applications, e.g. in error correcting codes, interpolation, engineering etc. From the Fundamental Theorem of Algebra we know that a polynomial of degree n with complex coefficients has n roots which may or may not be distinct. The problem of finding the roots of a given polynomial was known since the Sumerians, i.e. 3000 BC. Since then many different methods

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

of finding the roots approximation were proposed, e.g. Newton's method [Var02], Harmonic Mean Newton's method [Ard11], Whittaker's method [Var02], Halley's method [Ard11], Chebyshev's method [Var02], Traub-Ostrowski's method [Var02] etc.

Let us consider a polynomial $p \in \mathbb{C}[Z]$ and $\deg p \geq 2$ of the form:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \dots + a_1 z + a_0. \quad (1)$$

Now we define a sequence of functions $D_m : \mathbb{C} \rightarrow \mathbb{C}$ for all $z \in \mathbb{C}$ [Kal09]:

$$D_0(z) = 1, \quad D_m(z) = \det \begin{pmatrix} p'(z) & \frac{p''(z)}{2!} & \dots & \frac{p^{(m-1)}(z)}{(m-1)!} & \frac{p^{(m)}(z)}{m!} \\ p(z) & p'(z) & \ddots & \ddots & \frac{p^{(m-1)}(z)}{(m-1)!} \\ 0 & p(z) & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \frac{p''(z)}{2!} \\ 0 & 0 & \dots & p(z) & p'(z) \end{pmatrix} \quad (2)$$

for $m \geq 1$.

Using the D_m sequence we define a Basic Family $\{B_m\}_{m=2}^\infty$, where $B_m : \mathbb{C} \rightarrow \mathbb{C}$, in a following way [Kal09]:

$$\forall z \in \mathbb{C} \quad B_m(z) = z - p(z) \frac{D_{m-2}(z)}{D_{m-1}(z)}. \quad (3)$$

The Basic Family is a fundamental part of polynomiography. Let us see how the first three elements of the Basic Family look like:

$$B_2(z) = z - \frac{p(z)}{p'(z)}, \quad (4)$$

$$B_3(z) = z - \frac{2p'(z)p(z)}{2p'(z)^2 - p''(z)p(z)}, \quad (5)$$

$$B_4(z) = z - \frac{6p'(z)^2 p(z) - 3p''(z)p(z)^2}{p'''(z)p(z)^2 + 6p'(z)^3 - 6p''(z)p'(z)p(z)}. \quad (6)$$

As we look at those formulas we see that B_2 is formula used in Newton's root finding method, and B_3 is formula used in Halley's method. Moreover, we see that when m increases the formula for B_m becomes more and more complex. So we need an efficient algorithm for its computation. In [Kal10] Kalantari introduced such algorithm (Algorithm 1). To derive this algorithm he used the theory of symmetric functions.

Algorithm 2 presents a method of determining polynomiograph [Kal09]. In the algorithm for each point in the considered area $A \subset \mathbb{C}$ we iterate given element of the Basic Family (defined by $p \in \mathbb{C}[Z]$ and $m \geq 2$). If

Algorithm 1: $B_m(z)$ computation

Input: $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – polynomial, $m \geq 2$ – number for B_m , $z_0 \in \mathbb{C}$ – point for which we make the computations.

Output: $B_m(z_0)$.

```

1  $h[0] = 1$ 
2 for  $i = 0$  to  $m - 1$  do
3    $e[i] = p^{(i)}(z_0)/(i!p(z_0))$ 
4 for  $i = 1$  to  $m - 1$  do
5    $h[i] = \sum_{r=0}^{i-1} (-1)^{i-r-1} e[i-r]h[r]$ 
6  $B_m(z_0) = z_0 - h[m-2]/h[m-1]$ 

```

the modulus of the difference between two successive points in the iteration process is smaller than the given accuracy $\varepsilon > 0$ we assume that the generated sequence converge to a root of p and we stop iterating. If we reach the maximum number of iterations k we assume that the generated sequence do not converge to any root of p . At the end we give a colour to the considered point using the given colourmap and the iteration number at which we have left the while loop.

Algorithm 2: Polynomiograph computation

Input: $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – polynomial, $A \subset \mathbb{C}$ – area, k – number of iterations, ε – accuracy, $m \geq 2$ – number for B_m , $colours[0..k]$ – colourmap.

Output: Polynomiograph for the area A .

```

1 for  $z_0 \in A$  do
2    $i = 0$ 
3   while  $i \leq k$  do
4      $z_{i+1} = B_m(z_i)$ 
5     if  $|z_{i+1} - z_i| < \varepsilon$  then
6       break
7      $i = i + 1$ 
8   Print  $z_0$  with  $colours[i]$  colour

```

Examples of polynomiographs generated using Algorithm 2 for:

(a) $p(z) = z^3 - 1$, $A = [-3, 3]^2$, $k = 20$, $\varepsilon = 0.001$, $m = 2$,

(b) $p(z) = -2z^4 + z^3 + z^2 - 2z - 1$, $A = [1, 2] \times [-0.5, 0.5]$, $k = 20$, $\varepsilon = 0.001$, $m = 3$,

(c) $p(z) = z^4 + z^2 - 1$, $A = [-3, 3]^2$, $k = 20$, $\varepsilon = 0.001$, $m = 4$,

(d) $p(z) = z^3 - 3z + 3$, $A = [-3, 3]^2$, $k = 10$, $\varepsilon = 0.001$, $m = 2$

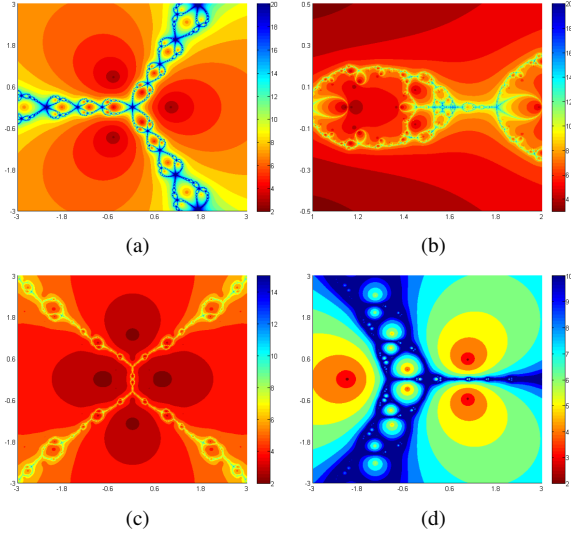


Figure 1: Examples of polynomiographs.

are presented in Fig. 1.

In Algorithm 2 to colour the points we use the iteration number for which we have left the iteration process, we call this method the iteration colouring. We can use different methods of colouring, e.g. basins of attraction (each polynomial root has its own colour, for each point in A we iterate it and when the condition in line 5 of Algorithm 2 is met the considered point gets the colour of the nearest root), mixed method (we mix the iteration colouring and the basins of attraction) etc. [Kal09].

3 DIFFERENT CONVERGENCE TESTS

In line 5 of Algorithm 2 we see a standard test for convergence of an iteration process in the numerical root finding methods. In the test we take two elements: the one computed in the current iteration and the element from the previous iteration, and we calculate the modulus of their difference. Then we check if the calculated value is smaller than the given accuracy. The modulus calculation in the test is equivalent to the computation of the distance between these two points of the complex plane. So we may change the way in which we calculate the distance with a different metric.

We know that the complex plane \mathbb{C} is isometric with \mathbb{R}^2 , where the isometry $\phi : \mathbb{C} \rightarrow \mathbb{R}^2$ is defined as follows [Sea07]:

$$\phi(z) = (\Re(z), \Im(z)) \quad (7)$$

for every $z \in \mathbb{C}$, and where $\Re(z)$, $\Im(z)$ denote the real and imaginary part of z (respectively). Using the isometry we can define metric $d : \mathbb{C} \times \mathbb{C} \rightarrow [0, +\infty)$ using metric $\rho : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow [0, +\infty)$ in a following way [Sea07]:

$$d(z_1, z_2) = \rho(\phi(z_1), \phi(z_2)), \quad (8)$$

where $z_1, z_2 \in \mathbb{C}$.

On \mathbb{R}^2 we have many different metrics which we may use [Sea07], e.g.

- taxicab metric

$$\rho((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|, \quad (9)$$

- supremum metric

$$\rho((x_1, y_1), (x_2, y_2)) = \max\{|x_1 - x_2|, |y_1 - y_2|\}, \quad (10)$$

- l_p metric

$$\rho((x_1, y_1), (x_2, y_2)) = [|x_1 - x_2|^p + |y_1 - y_2|^p]^{\frac{1}{p}}, \quad (11)$$

where $1 \leq p \leq +\infty$.

When we have some metric space (X, ρ) we can define new metrics using following facts [Sea07]:

- if $f : X \rightarrow X$ is injective, then

$$\eta(x, y) = \rho(f(x), f(y)) \quad (12)$$

is a metric on X ,

- if $f : X \rightarrow \mathbb{R}$ is a function, then

$$\eta(x, y) = \rho(x, y) + |f(x) - f(y)| \quad (13)$$

is a metric on X .

From the examples presented in the next section we will see that changing the metric produces only a small change in the shape of polynomiograph. As we are interested in generation of interesting patterns using the polynomiography and not in the best convergence of the numerical method we can relax the assumption about the metric. For this purpose we can take $p \in (0, 1)$ in the l_p metric obtaining the so-called fractional distance which is used for instance in models for forecasting pollution concentrations [DW12].

We also can omit the assumption about the injectivity of f in (12). For instance when we take \mathbb{C} with the modulus metric and $f(z) = |z|^2$, which is not injective, we obtain:

$$\eta(z_1, z_2) = ||z_1|^2 - |z_2|^2|. \quad (14)$$

The η function from (14) was used instead the modulus test by Pickover in Halley's method in [Pic88]. In this way Pickover obtained very diverse shapes of the polynomiographs.

Another way to modify the tests is to add some weights in the metric functions. When we use (12) we can add two weights $\alpha, \beta \in \mathbb{R}$ in a following way:

$$\eta(x, y) = \rho(\alpha f(x), \beta f(y)). \quad (15)$$

In this way we loose the metric property of η , e.g. it is not symmetric for $\alpha \neq \beta$, but as we will see in section 4 we obtain very diverse polynomiographs using this function.

Till now the proposed tests were based on metrics, but there is no obstacle in using tests which are based on functions that are not metric, quasimetrics etc. at all. For instance we can use following tests:

$$|\exp(\alpha z_{i+1} - \beta z_i)| < \varepsilon, \quad (16)$$

$$|\alpha \Re(z_{i+1} - z_i)| < \varepsilon \vee |\beta \Im(z_{i+1} - z_i)| < \varepsilon, \quad (17)$$

$$|\alpha \Re(z_{i+1} - z_i)|^2 < \varepsilon \wedge |\beta \Im(z_{i+1} - z_i)|^2 < \varepsilon, \quad (18)$$

where $\alpha, \beta \in \mathbb{R}$. In the tests which consist of several terms joined with logical operators, e.g. (17), (18), instead of one ε we can use separate value for each term.

The last group of tests which we propose is based on the idea taken from the escape time algorithm which is used in the Julia set drawing. Similar like in the escape time algorithm we can check if a value of some iterated function escapes, i.e. is greater than the given radius $R > 0$. Examples of this kind of tests are:

$$|z_{i+1} - z_i| + |\arg(z_{i+1}) - \arg(z_i)| > R, \quad (19)$$

$$\left| \frac{1}{|z_{i+1}|^2} - \frac{1}{|z_i|^2} \right| + ||z_{i+1}|^2 - |z_i|^2| > R, \quad (20)$$

$$\alpha |\Re(z_{i+1} - z_i)| > R \wedge \beta |\Im(z_{i+1} - z_i)| > R, \quad (21)$$

where $\arg(z)$ is an argument of complex number z , and $\alpha, \beta \in \mathbb{R}$.

4 EXAMPLES

In this section we show some examples of using the different tests proposed in section 3. We start our examples with changing the standard metric (modulus) used in the polynomiography with the supremum metric. In the example we use: $p(z) = z^3 - 3z + 3$, $A = [-2, 2]^2$, $k = 15$, $\varepsilon = 0.001$, $m = 2$. Figure 2(a) presents the result for the modulus metric and Fig. 2(c) presents the result for the supremum metric. From the figures we see that in both cases the result is very similar and the difference is small. To see the difference more precisely in Figs. 2(b), 2(d) magnification of the marked areas from Figs. 2(a), 2(c) are presented. In the case of modulus metric we have smooth boundaries between the regions and for the supremum metric the boundaries are frayed and the regions are lighter. When we use a different metric instead of the supremum metric the effect will be very similar, so the obtained results are not interesting from the artistic point of view.

In the next example we use the test used by Pickover (14) and its weighted modification. The common parameters used in the example: $p(z) = z^4 + z^2 - 1$, $A = [-3, 3]^2$, $k = 15$, $\varepsilon = 0.001$, $m = 2$. Figure 3(a) presents

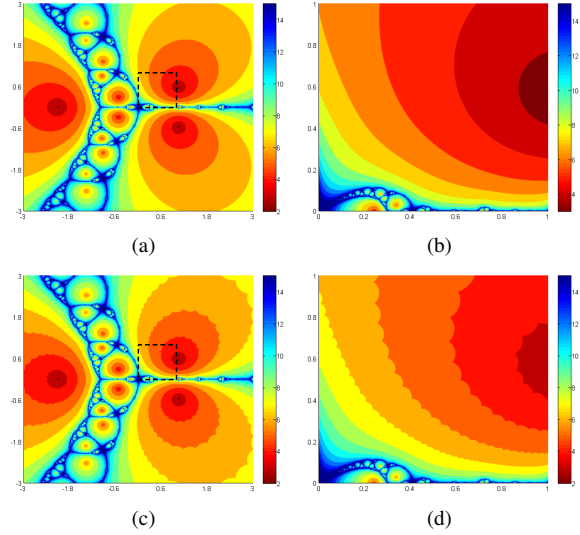


Figure 2: Examples of polynomiographs: (a) with modulus metric, (b) with supremum metric, (c) magnification of the marked area from (a), (d) magnification of the marked area from (c).

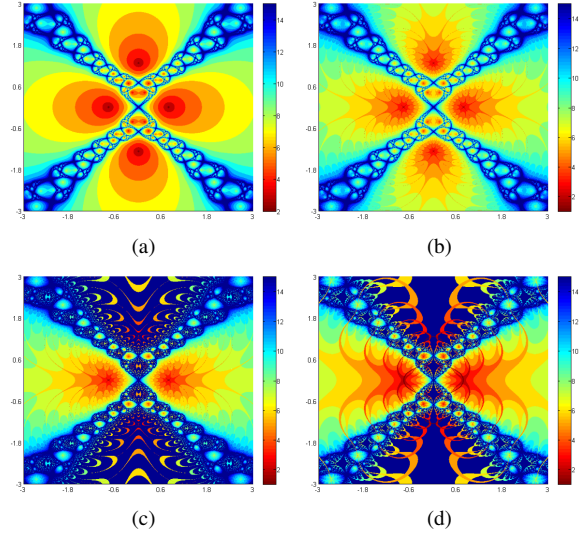


Figure 3: Examples of polynomiographs: (a) original, (b) using the Pickover test, (c), (d) using the weighted version of Pickover test.

the result for the original test, Fig. 3(b) for the Pickover test and Figs. 3(c), 3(d) the results for weighted version of (14), i.e. $|\alpha |z_1|^2 - \beta |z_2|^2|$, where $\alpha = 1.05$, $\beta = 1.049$ for (c) and $\alpha = 0.049$, $\beta = 0.05$ for (d).

The Pickover test changes the regions of polynomiograph where the convergence using the original test was fast. In this way we obtain some swirls in the smooth areas. Using the test with weights we obtain even more changes in the areas of the fast convergence and moreover small changes in the areas of the slow convergence. The polynomiographs obtained with the non-standard test look very interesting and the patterns are more complex comparing to the original one.

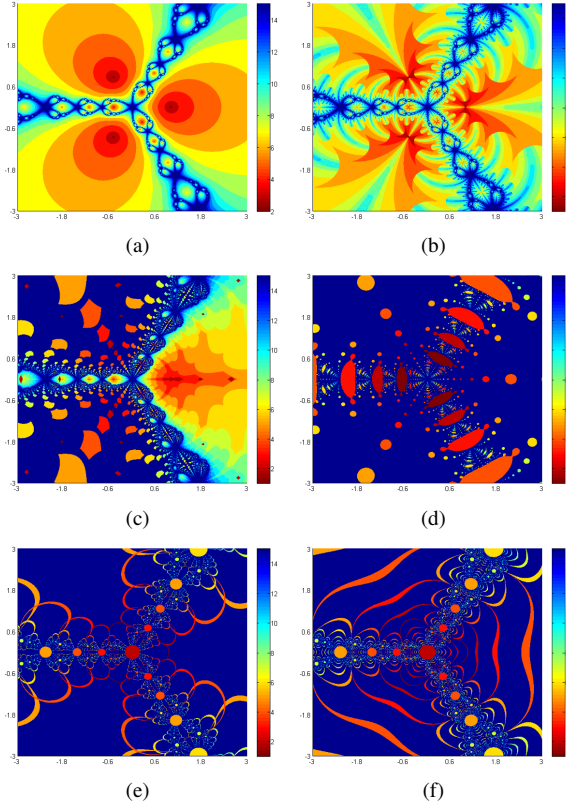


Figure 4: Examples of polynomiographs with different tests based on metrics and weights.

In the previous example we used only the Pickover test and now we show examples of more tests which are based on metrics and weights. The common parameters used in the example: $p(z) = z^3 - 1$, $A = [-3, 3]^2$, $k = 15$, $\varepsilon = 0.001$, $m = 2$. Figure 4(a) presents the original polynomiograph and Figs. 4(b)-(f) present polynomiographs obtained with the help of different metrics and weights. The tests used in the example were following:

- (a) $|z_{i+1} - z_i| < \varepsilon$,
- (b) $|0.01(z_{i+1} - z_i)| + |0.029|z_{i+1}|^2 - 0.03|z_i|^2| < \varepsilon$,
- (c) $|0.05 \sin(\Re(z_{i+1})) - 0.049 \sin(\Re(z_i))| + |0.05 \sin(\Im(z_{i+1})) - 0.049 \sin(\Im(z_i))| < \varepsilon$,
- (d) $|0.01z_{i+1}^{10} - 0.008z_i^{10}| < \varepsilon$,
- (e) $|\frac{0.05}{|z_{i+1}|^2} - \frac{0.045}{|z_i|^2}| < \varepsilon$,
- (f) $|\frac{0.045}{|z_{i+1}|^2} - \frac{0.05}{|z_i|^2}| < \varepsilon$.

From the presented polynomiographs we see that using the different metrics and weights we are able to obtain very diverse and interesting patterns comparing to the original test. In the Fig. 4(b) we can observe a pattern which looks like a knot and in Fig. 4(c) pattern which reminds a flower. From Fig. 4(e) and Fig. 4(f) we see

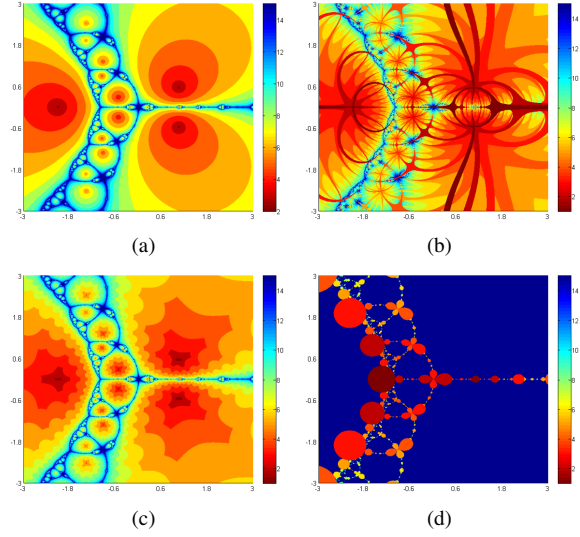


Figure 5: Examples of polynomiographs: (a) original, (b)-(d) based on the non-metric tests.

that the patterns look quite different, but the tests used for their creation differ only in order of the weights (they are interchanged).

Next example presents the use of the tests which are based on the non-metric conditions. The common parameters used in the example: $p(z) = z^3 - 3z + 3$, $A = [-3, 3]^2$, $k = 15$, $\varepsilon = 0.001$, $m = 2$. Figure 5(a) presents the original polynomiograph and Figs. 5(b)-(d) present polynomiographs obtained with the help of following tests:

- (b) $|0.04\Re(z_{i+1} - z_i)| < \varepsilon \vee |0.05\Im(z_{i+1} - z_i)|\varepsilon$,
- (c) $|0.49\Re(z_{i+1} - z_i)|^2 < \varepsilon \wedge |\Im(z_{i+1} - z_i)|^2 < \varepsilon$,
- (d) $|\exp(10z_{i+1} - 9z_i)| < \varepsilon$.

Also in this case we see that when we change the modulus test to the tests based on the non-metric conditions we obtain very interesting patterns. For instance in Fig. 5(b) we see a very complicated net of swirls and in Fig. 5(d) a pattern which looks like a necklace.

In the last example we show some polynomiographs obtained with the tests basing on the escape criteria. The common parameters used in the example: $p(z) = -2z^4 + z^3 + z^2 - 2z - 1$, $A = [1, 2] \times [-0.5, 0.5]$, $k = 15$, $m = 2$. Figure 6 presents the original polynomiograph for $\varepsilon = 0.001$ and Figs. 6(b)-(d) present polynomiographs obtained with the help of following tests: (b) $R = 6$ and condition (19), (c) $R = 8$ and condition (20), (d) $R = 6$ and condition (21) for $\alpha = 8$ and $\beta = 11$.

The patterns obtained with the escape criteria also differ from the original one. But obtaining a very interesting pattern using those criteria is difficult. This is because the patterns arise in the regions where the original method converges very slowly or reaches the maximum number of iterations.

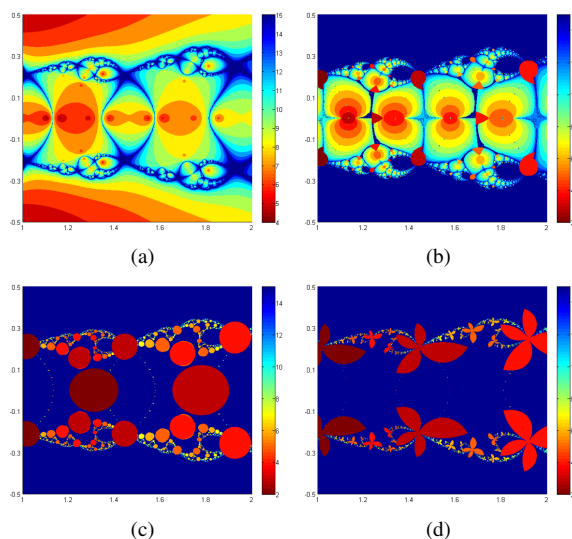


Figure 6: Examples of polynomiographs: (a) original, (b)-(d) based on the escape criteria.

5 CONCLUSIONS

In this paper we presented modifications of the polynomiography algorithm. The modifications were based on the change of the usual convergence test with the tests based on distance and non-distance conditions. Presented examples show that using the proposed tests we are able to obtain very interesting patterns. We believe that the results of this paper can enrich the functionality of the existing polynomiography software.

When we search for an interesting pattern using the polynomiography we must make the right choice of a polynomial, the iteration function etc. and using the trial and error we must find an interesting area [Kal09]. Adding our tests to the list of polynomiography parameters we make the search even more difficult, so there is a need for automatic method which finds interesting patterns. The notion of an interesting pattern is very difficult to define and usually is based on a subjective feeling, but there are some attempts to estimate the notion. Ashlock and Jamieson in [AJ08] introduced a method of exploring the Mandelbrot and Julia sets for interesting patterns. They used evolutionary algorithms with different fitness functions. In our further research we will try to develop a method which searches for interesting patterns in the polynomiography using similar methodology like that presented by Ashlock.

Polynomiography is based on the complex polynomials. In [Lev94] we can find examples of using q -systems numbers instead of complex numbers for obtaining diverse patterns, and in [WS13] we find bicomplex numbers used in the Mandelbrot and Julia sets. Using the q -system and bicomplex numbers in the poly-

nomiography can probably further enrich the patterns obtained with the polynomiography.

6 REFERENCES

- [Ard11] Ardelean, G.: A Comparison Between Iterative Methods by Using the Basins of Attraction. *Applied Mathematics and Computation* 218(1), 88-95, (2011)
- [AJ08] Ashlock, D., Jamieson, B.: Evolutionary Exploration of Complex Fractals. In: P.F.Hingston, L.C.Barone, Z.Michalewicz (eds.) *Design by Evolution*. Springer, Berlin, pp. 121-143, (2008)
- [DW12] Domańska, D., Wojtylak, M.: Application of Fuzzy Time Series Models for Forecasting Pollution Concentrations. *Expert Systems with Applications* 39(9), 7693-7679, (2012)
- [Kal04] Kalantari, B.: Polynomiography and Applications in Art, Education and Science. *Computers & Graphics* 28(3), 417-430, (2004)
- [Kal05a] Kalantari, B.: Method of Creating Graphical Works Based on Polynomials. U.S. Patent 6,894,705, issued May 17, 2005
- [Kal05b] Kalantari, B.: Two and Three-dimensional Art Inspired by Polynomiography. *Proceedings of Bridges, Banff, Canada*, pp. 321-328, (2005)
- [Kal09] Kalantari, B.: *Polynomial Root-Finding and Polynomiography*. World Scientific, Singapore (2009)
- [Kal10] Kalantari, B.: A Combinatorial Construction of High Order Algorithms for Finding Polynomial Roots of Known Multiplicity. *Proceedings of the American Mathematical Society* 138(6), 1897-1906, (2010)
- [Lev94] Levin, M.: Discontinuous and Alternate Q -System Fractals. *Computer & Graphics* 18(6), 873-884, (1994)
- [Pic88] Pickover, C.A.: A Note on Chaos and Halley's Method. *Communications of the ACM* 31(11), 1326-1329, (1988)
- [Pic01] Pickover, C.A.: *Computers, Pattern, Chaos, and Beauty: Graphics from an Unseen World*. Dover Publications, Mineola, (2001)
- [Sea07] Searcoid, M.Ó: *Metric Spaces*. Springer, London, (2007)
- [Var02] Varona, J.L.: Graphics and Numerical Comparison Between Iterative Methods. *The Mathematical Intelligencer* 24(1), 37-46, (2002)
- [WS13] Wang, X.-Y., Song, W.-J.: The Generalized M-J Sets for Bicomplex Numbers. *Nonlinear Dynamics* 72(1-2), 17-26, (2013)

Fast Normal Approximation of Point Clouds in Screen Space

Daniel Schiffner
Goethe Universität
Robert-Mayer-Strasse 10
D-60054 Frankfurt
dschiffner@gdv.cs.uni-frankfurt.de

Marcel Ritter
University of Innsbruck &
Airborne Hydromapping OG
Technikerstr. 13a & 21
A-6020, Innsbruck, Austria
marcel.ritter@uibk.ac.at

Werner Benger
Center for Computation and
Technology,
Louisiana State University
216 Johnston Hall
LA 70803, Baton Rouge, USA
werner@cct.lsu.edu

ABSTRACT

Displaying large point clouds of mainly planar point distributions yet comes with large restrictions regarding the surface normal and surface reconstruction. Point data needs to be clustered or traversed to extract a local neighborhood which is necessary to retrieve surface information. We propose using the rendering pipeline to circumvent a pre-computation of the neighborhood in world space to perform a fast approximation of the surface in screen space. We present and compare three different methods for surface reconstruction within a post-process. These methods range from simple approximations to the definition of a tensor surface. All these methods are designed to run at interactive frame-rates. We also present a correction method to increase reconstruction quality, while preserving interactive frame-rates. Our results indicate, that the on-the-fly computation of surface normals is not a limiting factor on modern GPUs. As the surface information is generated during the post-process, only the target display size is the limiting factor. The performance is independent of the point cloud's size.

Keywords

Normal Reconstruction, Tensor Information, GPU, Point Clouds

1 INTRODUCTION

Huge data sets are nowadays generated by simulations or by observational methods. Point clouds are e.g. the result of particle based simulation codes or laser scans, such as airborne light detection and ranging (LIDAR) scanning. Surface related information, such as the surface normal, can be used to enhance the visualization of point clouds, e.g. for illumination. Traditional methods for reconstruction surface information require an expensive spatial sort operation. Therefore, these are executed during a pre-process. Our method aims at improving the exploration of LIDAR data sets, before applying more expensive approaches.

In our work, we use the large data throughput of modern GPUs to generate a fast estimation of the surface properties within screen space. Therefore, we apply three possible approaches and compare the individual results. The first approach uses the fragment shader specific

$dFdx$ and $dFdy$ functions. The second method calculates the surface normal by computing the cross product in a local neighborhood, which is available through the pixel neighborhood. The third applies a moving-least-squares approach to acquire tensor information. The resulting co-variance matrix is then used to compute the eigenvalues and eigenvectors.

In the next section, we list similar methods to our approach. Then, we present our methods and solutions to encountered issues. These methods are compared to each other and some examples are presented. Finally, we conclude with a summary of our findings and an outlook regarding future work.

2 RELATED WORK

Generic visualization frameworks, such as openWalnut [Walnut] or the visualization shell (VISH) are utilized for data exploration and processing of a large data sets. More expensive approaches to compute visual enhancements of points distributed on surfaces and lines, and geometrical reconstructions of lines have been done in [Bou212], [Rit12b] or [Rit12a].

The calculation of a surface normal is strongly connected to any surface reconstruction method. Especially for point based representations, methods using co-variance techniques [Ber94][Bjö05] are well

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

suited, because no exact neighborhood is available and some noise is to be expected. Alexa defined the so-called point-set surfaces and presented some projection specific calculations [Ale04]. The co-variance matrix allows to assess the quality of the point cloud data set using direct tensor field visualization methods, such as displaying tensor splats [Ben04]. To compute the eigenvalues and eigenvectors from a given co-variance matrix, the analytical approach presented by Hasan [Has01] or one of the methods presented by Kopp [Kop06] can be applied.

Yet, these methods rely on the identification of an accurate neighborhood. To acquire this information, the input data set needs to be sorted. Neighbors are either found by a brute force approach – which is not suitable at all –, by a tree search or by a Morton ordering [Con10]. A tree as well as a Morton order are highly suited for parallelization.

Instead of creating a kd-tree or a Morton order in world space, a neighborhood can also be computed in screen space. Thus, the computation is only performed on the currently visible part of the data set. This is commonly done by splatting the data points and extracting the properties from the frame buffer. Similar to the approach presented by [Sch11] or [Yan06], we use only screen space information for the selection of the neighborhood. The splats are projected using either a fixed or adapted point size, as proposed by Rusinkiewicz [Rus00]. Once the surface information is available, also high quality splatting techniques [Bot05] could be applied.

3 APPROACH

We use the information available in screen space to reconstruct a surface and its corresponding normals. We designed an approach consisting of three individual steps, as illustrated in figure 1.

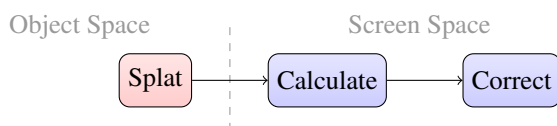


Figure 1: The outline of our screen space normal reconstruction. The first pass consists of splatting the depth values which are used in the consecutive passes. The second pass approximates the surface normal, while the optional third pass smooths the resulting values.

The first pass is a simple splatting of the input data and provides the depth information required by reconstruction. Each pixel is hereby surrounded by neighbor candidates. The second pass uses these depth values and computes surface properties. The candidates are inspected and rejected if the distance is too large, i.e. their interpolation weight is too small. The last pass is

optional and allows a further enhancement of the quality of the reconstructed properties.

Splatting the Point Cloud

We draw the point cloud, which will be reconstructed, using either a fixed or an approximate point size. Our approach only requires a depth buffer for computation of the surface information. As the depth-buffer is generated, in general, by all rendering approaches, this method can be applied to all scenarios.

To increase the accuracy, we encourage using a multi-sample depth-buffer. This allows the retrieval of multiple depth values per individual sample. Using a sampling count of 8 means that we are able to capture – at most – 8 individual splat depth values at once. It is, of course, possible that the unprojected world space coordinates are identical or invalid, i.e. the depth value was not set. Still it increases the stability of the following normal calculations. Multi-sampling is only applied within the first post process.

Normal Definition

We calculate the world space coordinates of the current pixel by un-projecting it based on the multi-sampled depth-buffer. The reconstruction of the surface normal can then be performed in three ways. The first method uses the local derivatives directly available in the fragment shader. The second and third method approximate the surface using a generic neighborhood description.

This neighborhood is defined by fixed sampling patterns. The most simple version takes 5 samples within a 3x3 neighborhood, while the most complex version selects 25 samples in a 7x7 neighborhood, see figure 4. The samples are focused on the diagonals, which increase the overall area captured during reconstruction. Note, that we use ascending indices for the opposite sample positions. This enables a simple definition of diagonals within a shader.

In our test, we did not observe any differences between the 5 and 9 sample schemes. This indicates, that the reduced representation is already able to capture the surface properties. The extended schemes, i.e. 17 and 25 samples, further increase stability of the results and are more comparable to off-line methods.

We orient all normals by inverting those, where the z-component is negative. All selected splat samples are visible and, thus, require a normal which is facing towards the camera.

To assure correct identification of possible neighbor candidates, a maximal distance is introduced. Neighboring pixels may not be true neighbors within world space due to the projection. Therefore, we reject every sample that is not within this configurable distance. This is comparable with the maximal distance in the MLS [Ale04] or tensor computations [Rit12a].

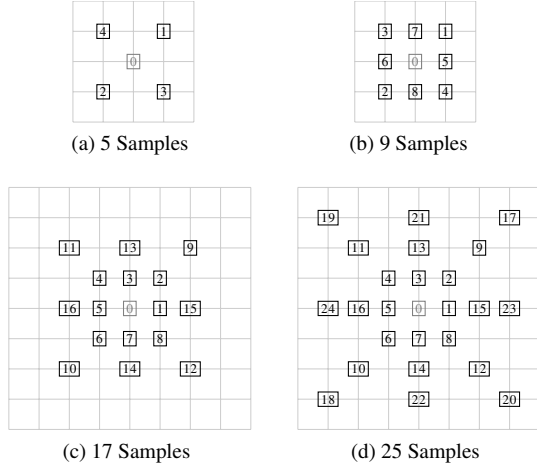


Figure 2: The used sampling schemes for defining the local neighborhood of a fragment. The center point 0 is optional.

Local Derivatives

Shaders support the calculation of local derivatives within the fragment shader since GLSL version 1.10. For reconstruction of the surface normal, the functions `dFdx` and `dFdy` are used. These internally extract neighbor positions from concurrent thread blocks and are only available in the fragment stage. This means that the surface is completely splatted and the individual samples may have overlapped. With c , the current position in clip-coordinates, the surface normal \vec{n} is computed:

$$\vec{n}(c) = dFdy(c) \times dFdx(c)$$

This method is very sensitive to noise or irregularities in the depth buffer and in many cases produces normals not representing a good reconstructed surface. However, if the surface is continuous and the splat size is carefully chosen, this method will suffice.

Plane Approximation

Similar to the computation of mesh surface properties, we approximate face normals within this approach. The normals are accumulated and the resulting vector is normalized. Finally, we impose an orientation and align the vector.

To obtain the needed vectors, we use one of the proposed sampling schemes. Each direction vector is built up either by diagonal or counter-clock-wise (ccw) samples. The diagonals generate smoother results and do not require the center point at sample 0. This is similar to the anti-alias algorithms used in the rendering pipeline. The ccw approach accounts more for local changes and takes the center point into account. In the diagonal case, we obtain the surface normal by using the following formula:

$$\vec{n}(c) = \frac{1}{N} \sum_{i=0}^{\lfloor \frac{N}{4} \rfloor} \vec{d}_{4i} \times \vec{d}_{4i+2}$$

With $\vec{d}_i = s_i - s_{i+1}$. We optimize the sampling schemes for a diagonal pattern, since we intend to create smooth surface normals with minimal noise.

Tensor Information

Using tensor information instead of flat patches leads to a smoother reconstruction. To derive this information, the computation of eigenvalues and eigenvectors is mandatory. We compute the point distribution tensor by deriving the co-variance matrix for the current position c , as presented by [Rit12a] and similar to [Bjö05]:

$$CM(c) = \frac{1}{N} \sum_{k=1}^N w_{ik} (d_{ik} \otimes d_{ik}^T)$$

where $d_{ik} = c - S_k$, d_{ik}^T is the transpose, N is the number of samples around center point c , S_k the sample and w_{ik} is a weighting function. Here, we apply a weighting of $w_{ik} = \frac{1}{\|d_{ik}\|^2}$.

The tensor product \otimes is built by the direction vectors pointing from the current fragment's world coordinate to its points in the neighborhood. The weighted sum of these vectors result in the final point distribution tensor.

We compute the eigenvalues with the "Cordano" method presented by [Kop06]. This approach results in more stable vectors than the method proposed by Hasan et al. [Has01]. Similar findings were made by the developers of openWalnut [Walnut]. The eigenvector related to the minor eigenvalue hereby represents the surface normal. The vector is easily oriented, since the calculation is performed in clip-coordinates and the normal vectors have to face the camera.

Smoothing Normals

In a second, optional, screen space pass we correct the computed normals. We extract and scale adjacent normals within a local neighborhood, where the center normal is being favored. The surface normal is yield by accumulation of the weighted vectors.

Different weights and neighborhood sizes can increase the accuracy of the result. However, this does not apply to all situations. Especially, when using the plane approximation method, quality decreases, when the normals contain lots of noise.

4 RESULTS

We implemented a prototype, which has been tested on a i5 670 system with 8 GB RAM and a GeForce 680 running on Windows 7.

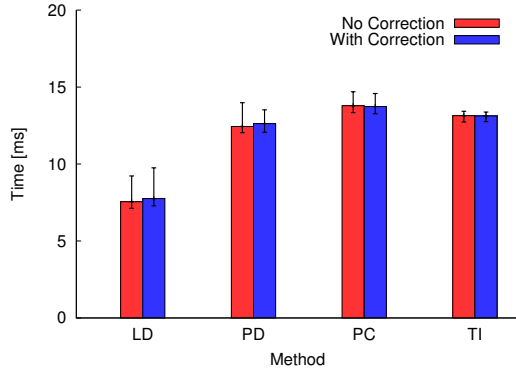


Figure 3: Timing results achieved using a screen size of 1024x768 with 8 multi-samples and the 9 samples scheme. LD denotes the local derivatives, PD the plane approximation using diagonals, PC the plane approximation using counter-clock-wise pattern, and TI the tensor information.

Timings

On all systems, we observed interactive frame rates with all methods. The fastest method is the local derivatives (LD) approximation, while the tensor information (TI) is the most expensive variant. The plane approximation with diagonals (PD) is slightly faster than the tensor variant. The ccw plane approximation (PC) is worse in terms of performance compared to the PD, due to the definition of the sampling scheme.

In figure 3, the average processing times are shown, including the generation of the depth values. We used a fixed multi-sampling count of 8 in all presented timing results. Thus, the real number of samples taken per pixel needs to be multiplied by 8. For better readability, we continue to use the introduced sampling count.

The splatting of the point cloud requires a significant amount of time. In our tests, it varied in the range of 30% to 50% mainly depend on the used screen and splat sizes.

The used sampling scheme size has a large influence on the performance and quality of the reconstruction, as seen in figure 4. The performance scales linearly with the number of used samples. However, the quality of the reconstruction is not necessarily improved when using a very high sampling count. This is due to the fact that the surface is smoothed and local information is suppressed.

We also measured the contribution of the individual steps performed by our approach. Interestingly, the splatting itself consumes a large amount of the overall processing time, while the correction requires only very little processing time. The larger the number of used samples, the higher the reconstruction times. Table 1 lists the detailed timings of the involved steps: “Splat” represents the splatting of the depth values, “Normal”

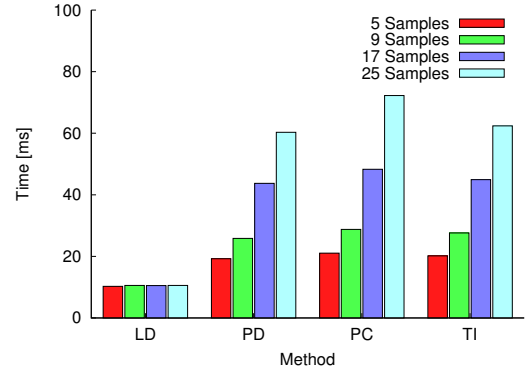


Figure 4: Influence of changing sampling scheme size for the reconstruction methods. Results taken with a screen resolution of 1600x1200. All methods use a 8 times multi-sampled depth-buffer.

9 Samples Scheme / 8 Multi-samples			
Operation	Min [ms]	Max [ms]	Avg [ms]
Splat	8.963	9.030	9.000
Normal	17.521	18.435	17.968
Correction	0.468	0.717	0.493
17 Samples Scheme / 8 Multi-samples			
Operation	Min [ms]	Max [ms]	Avg [ms]
Splat	8.801	10.654	8.980
Normal	34.278	35.711	34.890
Correction	0.466	5.740	0.702

Table 1: Distribution of the processing times among the individual operations of the proposed method. Results taken with a screen resolution of 1600x1200 using the tensor method.

the reconstruction and “Correction” the final smoothing.

Visual Results

All methods are able to reconstruct both noisy and smooth surfaces. We use several splatted object point clouds as test cases. All point clouds consist of at least 250k points to assure a high sampling density.

The results of the described reconstruction methods are shown in figure 5. These indicate that the TI method provides a stable and accurate reconstruction. The PD approach provides excellent results in smooth data sets. The LD approach always generates large noise. Despite not being suitable for a high quality surface approximation, it is the fastest approach.

To simulate noisy data, we alter the vertex positions within the splat shader. A light source is positioned below the object. The illuminated scene is shown in figure 6. The TI method generates the smoothest result, while the PD method yields more normals that differ widely from the original ones. The LD method provides the worst reconstruction. All methods generate

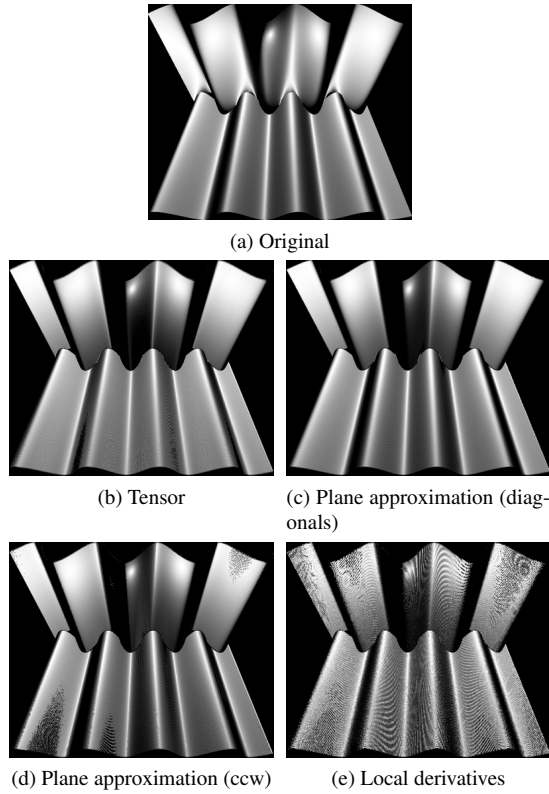


Figure 5: Reconstruction of the surface normal used for illumination. (a) shows the original object with pre-computed normals. (b) to (e) depict the proposed reconstruction methods.

more invalid normals in the low sampled region on the top.

Figure 7 illustrates the influence of the optional correction pass. The corrected normals are smoother and the number of correctly oriented surface normals is higher. The vectors are visualized via colors showing the x-, y-, and z-coordinates as red, green, and blue values.

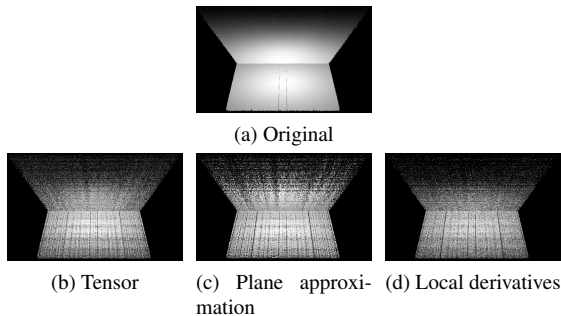


Figure 6: Reconstructed normals used for illumination in a test scenario with two planes. Noise is added to the input data. Even normals at the edge are well reconstructed, but tend to be smoothed.

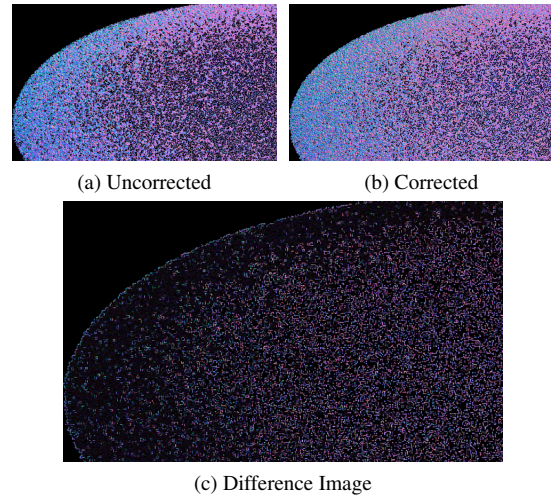


Figure 7: The influence of the correction pass applied to an ellipsoidal surface. The surface xyz-normal is illustrated as a rgb-color. The corrected version (b) contains more valid normals. The difference is visualized in (c).

Since the correction pass is very fast and increases the stability of the reconstruction, we always enable this pass in the following tests.

Application to a LIDAR Data Set

A point cloud stemming from an airborne laser-scan is used for further investigation of the technique and validation of the technique by a real-world application. We chose a small section of a bathymetric scan of the river Loisach in Bavaria (Germany), acquired with the hydrographic laser scanner Riegl VQ-820G [Ste10]. The scan contains different kinds of structures: fields, trees, lower vegetation, a river, a street with cars, power cables and a steep slope partially covered with vegetation. Figure 8 shows a side and a top view of the scan.

The two million points are colored by the minor eigenvector of the point distribution tensor computed in world-space.

The point distribution tensor was computed by using a neighborhood radius of 0.5, 1.0 and 2.0 meters. Two different weighting functions have been tested: constant weight and $\frac{1}{\|d_{ik}\|^2}$ weight. Using a kd-tree for finding neighbors and 6 OpenMP parallel threads on an Intel Xeon X560@2.67GHz the according computation times are 41, 85, and 218 seconds for the three radii. This computation of the tensor is a demanding computational tasks. However, it has been shown, that the tensor can be used for feature extraction, object recognition, and to improve the segmentation of point clouds [Rit12a][Rit12b][Bjö05]. When just looking at the minor eigenvector via color, the fields, the river, the street, the slope and the vegetation can be well distinguished from each other, visually.

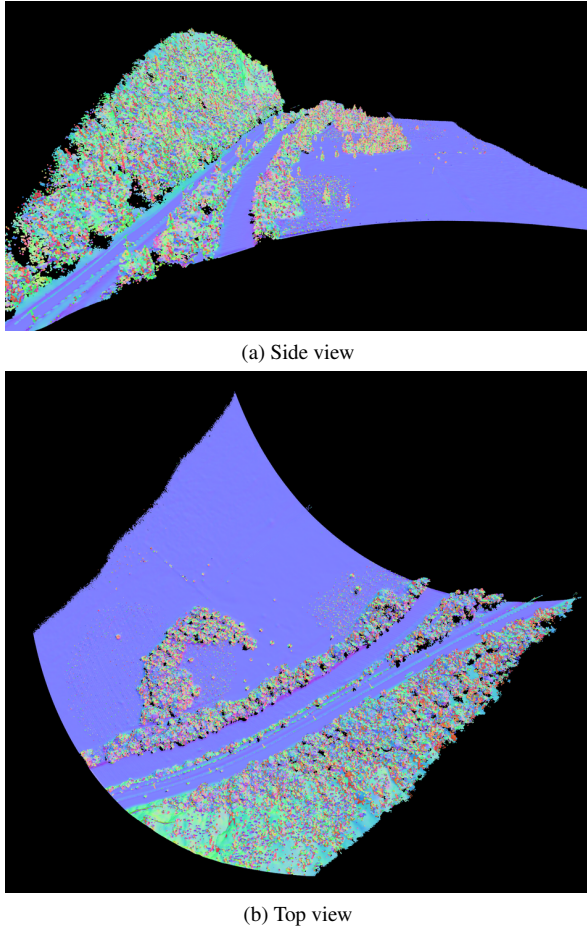


Figure 8: LIDAR laser-scan of a section of the Bavarian river Loisach in Germany. Laser echoes are illustrated as colored points. Color shows the minor eigenvector of the point distribution tensor. Vegetation can be visually distinguished from the ground and the river.

Next, we compare this expensive, fine grain computation in world space with our screen space technique. The results indicate that the approach is able to reconstruct the normals with rather high quality. The normals widely match with the normals calculated in world space, as shown in figure 9. However, differences in the forest areas of the scan are visible.

Also, where the sampling density near the camera position is not high enough to ensure high quality reconstruction in this region.

To compare the results of the different methods, we recorded a series of images from the Loisach data set. The TI method produces the most reliable results, while requiring a high sampling count. The PD method is able to create very smooth normals regardless of small surface changes, e.g. the missing power line in the upper region 10. The PC method includes it, but is more unstable. The LD method is the most efficient approach while yielding the worst quality in comparison to the other methods.

The correction pass increases the quality and the stability of the results by reducing the number of invalid surface normals. Figure 10c illustrates the enabled correction pass and figure 10d.

5 CONCLUSION

Our results show that a fast approximation of the surface normal can be achieved in real-time. Here, the surface is solely reconstructed from the depth-buffer and projection parameters. With our approach a preprocessing of surface information may be delayed until a region of interest has been selected. The results indicate that especially the tensor-based approach to determine the surface normal of a point cloud is a well-working method.

In comparison to the off-line world space method, we are able to create similar results at interactive frame rates. The loss of quality is negligible and is only visible in under-sampled regions. However, this method can only provide an approximation of the real point-cloud's surface information. The tests show that an increase of the neighborhood size decreases the performance linearly. A good quality is already achieved with small neighborhood sizes. The focus on the diagonals in the sampling schemes reduce the number of required samples.

6 FUTURE WORK

We plan to combine this technique with level of detail rendering to provide good visual representations of large airborne LIDAR scans. The surface normals provide important information to control such a level of detail algorithm.

The splatting technique could be enhanced by utilizing more information represented in the point distribution tensor. Extracting some features of the tensor will improve the readability of point clouds without expensive pre-computations.

Additionally, we plan to enhance the reconstruction method by providing more weighting functions besides the $\frac{1}{\|d_{ik}\|^2}$ weight for the computation of the co-variance matrix.

To avoid expensive re-calculations, we plan to employ a caching strategy. A re-computation of the surface normals would only be required when camera location or point coordinates are changed, further increasing the overall performance of the approach.

7 ACKNOWLEDGEMENTS

Special thanks to Frank Steinbacher for providing the LIDAR data set of the river Loisach. This work was supported by the Austrian Science Foundation FWF DK+ project Computational Interdisciplinary Modeling

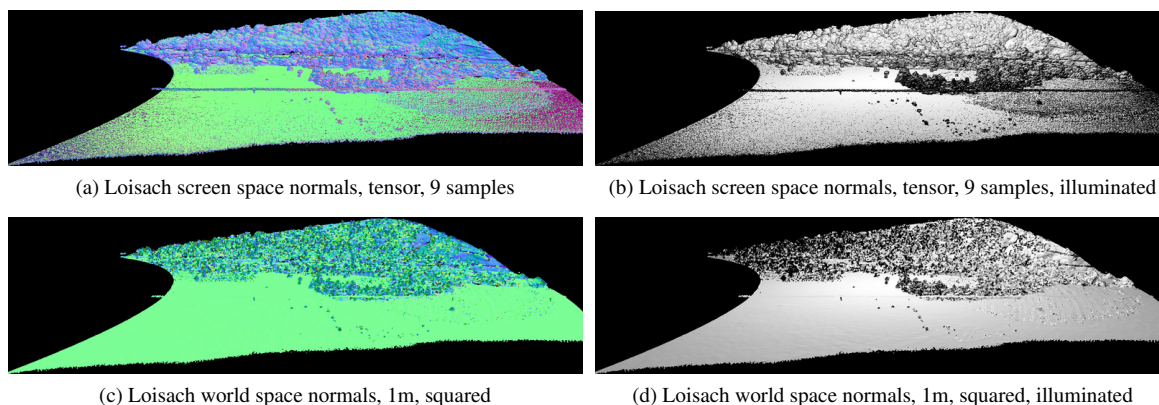


Figure 9: The reconstruction of the minor eigenvector using the fast screen space approach.

(W1227), and grant P19300. This research employed resources of the Center for Computation and Technology at Louisiana State University, which is supported by funding from the Louisiana legislatures Information Technology Initiative. This work was supported by the Austrian Ministry of Science BMWF as part of the Uni-Infrastrukturprogramm of the Forschungsplattform Scientific Computing at LFU Innsbruck.

8 REFERENCES

- [Con10] Connor, M., and Kumar, P.: Fast Construction of k-Nearest Neighbor Graphs for Point Clouds. *IEEE TVCG* 16, No.4. pp.599–608, 2010.
- [Yan06] Yang, R., Guinip, D., Wang, L.: View-dependent textured splatting. *The Visual Computer* 22, pp.456–467, 2006.
- [Has01] Hasan, K.M., Bassier, P.J., Parker, D.L., Alexander, A.L.: Analytical computation of the eigenvalues and eigenvectors in DT-MRI. *J. Magn. Reson.* 152, pp.41–47, 2001.
- [Ale04] Alexa, M., Rusinkiewicz, S., and Adamson, A.: On normals and projection operators for surfaces defined by point sets. *Eurographics Symp. PBG.*, pp. 149–155, 2004.
- [Bou12] Boulch, A., and Marlet, R.: Fast and Robust Normal Estimation for Point Clouds with Sharp Features. *Comp. Graph. Forum* 31, No.5, pp.1765–1774, 2012.
- [Walnut] Open Walnut. <http://openwalnut.org>.
- [Ben07] Benger, W., Ritter, G., Heinzl, R.: The Concepts of VISH. 4th High-End Vis. Workshop, pp.26–39, 2007.
- [Ben04] Benger, W., Hege, H.-C.: Tensor splats. *Conf. on Vis. and Data Analysis*, Vol.5295, pp.151–162, 2004.
- [Ber94] Berkmann, J., and Caelli, T.: Computation of surface geometry and segmentation using covariance techniques. *IEEE TPAMI* 16, No.11, pp.1114–1116, 1994.
- [Rit12a] Ritter, M., Benger, W., Cosenza, B., Pullman, K., Moritsch, H., Leimer, W.: Visual Data Mining Using the Point Distribution Tensor. *IARIS Workshop on Computer Vision and Computer Graphics, VisGra*, 2012.
- [Rit12b] Ritter, M., Benger, W.: Reconstruction Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. *WSCG*, pp.223–230, 2012.
- [Bjö05] Johansson, B., and Moe, A.: Object Recognition in 3D Laser Radar Data using Plane triplets, technical report LiTH-ISY-R-2708, Dept. EE, Linköping University, 2005.
- [Rus00] Rusinkiewicz, S., Levoy, M.: Qsplat: A Multiresolution Point Rendering System for Large Meshes, *SIGGRAPH '00*, pp.343–352, 2000.
- [Bot05] Botsch, M., and Hornung, A., and Zwicker, M., and Kobbelt, L.: High-quality surface splatting on today's GPUs. *Eurographics VGTC Symposium on PBG*, pp.17–24, 2005.
- [Sch11] Schiffner, D., Krömker, D.: Three Dimensional Saliency Calculation Using Splatting, 6th ICIG, pp.835–840, 2011.
- [Shi09] Shirley, P., and Marschner, S.: *Fundamentals of Computer Graphics*, 3rd Edition, A.K. Peters Ltd, 2009.
- [Kop06] Kopp, J.: Efficient numerical diagonalization of hermitian 3x3 matrices, *arXiv:physics/0610206v1*, 2006.
- [Ste10] Steinbacher, F., Pfennigbauer, M., Ulrich, A., and Aufleger, M.: Vermessung der Gewässersohle - aus der Luft - durch das Wasser, in *Wasserbau in Bewegung ... von der Statik zur Dynamik. Beiträge zum 15. Gemeinschaftssymposium der Wasserbau Institute TU München, TU Graz und ETH Zürich*, 2010.

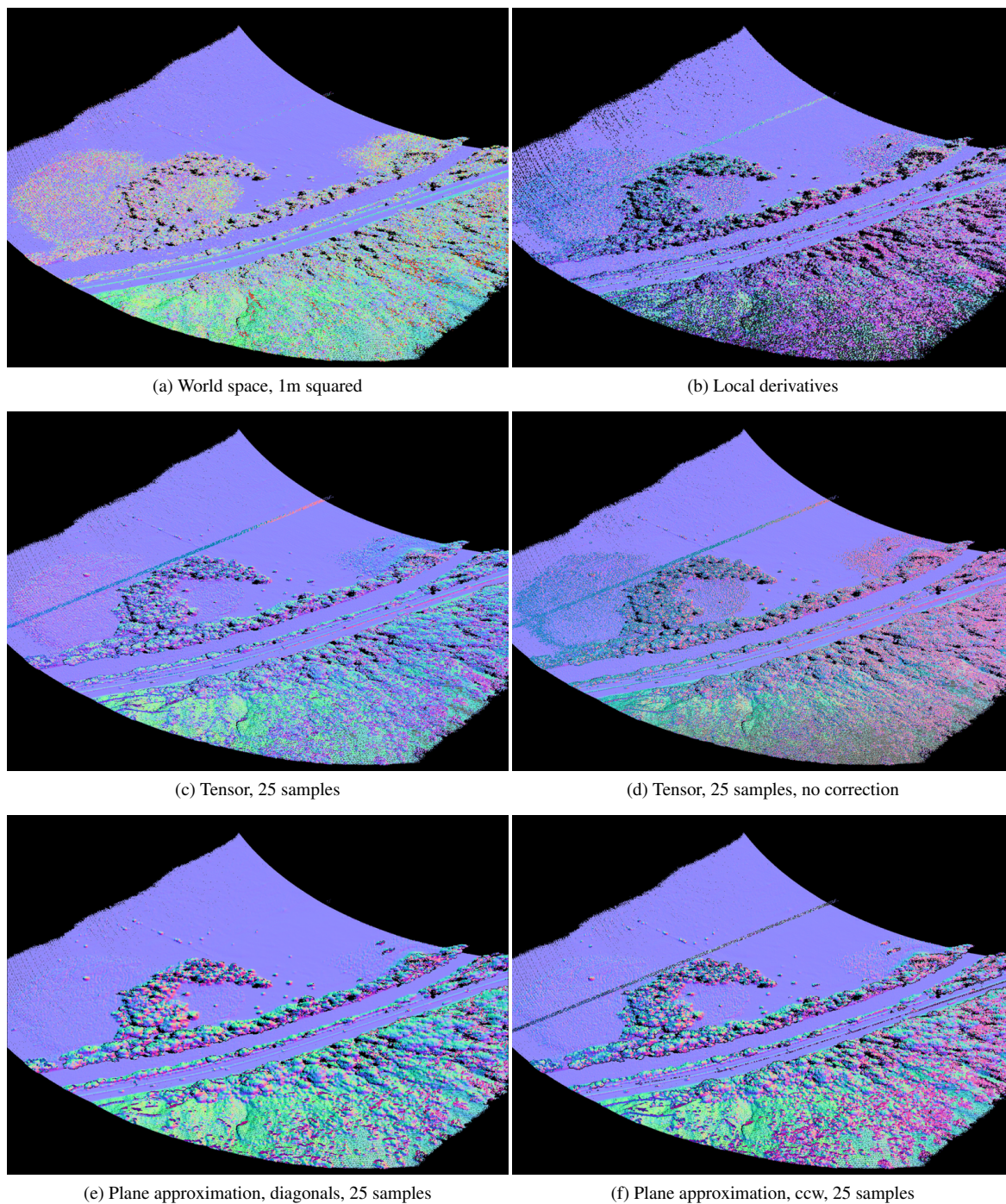


Figure 10: Comparison of the different reconstruction methods used on the Loisach LIDAR data set.

Using OpenGL State History for Graphics Debugging

Bryce van Dyk, Christof Lutteroth, Gerald Weber, Burkhard Wünsche

Department of Computer Science

University of Auckland

Private Bag 92019, Auckland 1142

New Zealand

bvan036@auckland.ac.nz, {christof, gerald, burkhard}@cs.auckland.ac.nz

ABSTRACT

To fulfill the unique debugging requirements of graphics programming, specialized tools are needed to aid in the debugging process. Modern graphics debuggers allow developers to inspect the current graphics state of a running application, and influence their control flow. However, they do not make maximum use of information about previous graphics states, despite the possible utility of this information in debugging. We propose GLDebug, an OpenGL debugger with novel features for using historical information to assist with graphics debugging. GLDebug provides the ability to capture and recall OpenGL state and function call information. Developers can retrace the graphics state history of OpenGL applications and compare different recorded states, which may come from different applications. State differences are made clearly visible, so that the source of state-based errors can be tracked down more easily. GLDebug was evaluated in a user study, with promising results: the participants found the tool helped them when working on four different OpenGL debugging tasks. All participants commented favorably on the support for tracking and analyzing state history. The results indicate that historical information is useful for graphics debugging, and that debuggers supporting such information can improve debugging efficacy.

Keywords

debugging, state history, function call history

1 INTRODUCTION

Computer graphics is applied in a vast number of fields such as entertainment, medicine, and computer-aided design. With so many applications for computer graphics, there is a demand for tools that assist programmers with the analysis and debugging of graphics code. However, general purpose debuggers do not cater to the specific needs of graphics programmers.

The need for dedicated tools stems from the unique paradigms used in graphics programming, as well as limitations due to the graphics hardware. For example, when programming with OpenGL, programmers must manage the state of OpenGL, treating OpenGL as a state machine. General purpose debuggers do not offer the ability to monitor this state – a useful feature that graphics debuggers should offer. Similarly, general purpose debuggers cannot help inspect the internal state of the graphics hardware – not in the same way they do for programs running on the CPU. There are also important differences in the types of data being dealt with: graphics debuggers must consider objects such as textures and matrices, which are of particular importance in graphics programming.

Various graphics debuggers have been introduced over the last decade by commercial vendors, open-source developers and researchers. These debuggers address the problems of inspecting the internals of graphics hard-

ware, controlling the execution flow of graphics code, and profiling it. However, their focus is on giving developers access to the current state of the graphics hardware only.

In this paper, we explore the idea of using historical information to assist with graphics debugging. We present a novel debugger, GLDebug, which provides the novel ability to capture and recall past OpenGL state and function call information. GLDebug allows developers to accumulate this historical information over time from multiple OpenGL applications, and compare it in a user interface that is similar to other history viewers. Users of GLDebug can retrace the graphics state history of OpenGL applications and compare different recorded states, making state changes clearly visible. This makes it easier, for example, to find defects in erroneous code when comparing it with working code. In particular, we are addressing the following research questions:

R1 How can graphics state history be supported in a debugger and presented to the user?

R2 In how far does the use of graphics state history facilitate debugging?

The ability to record and inspect graphics API states has been discussed in prior work [3, 5], so we only give a brief overview of this. In particular, we point out the

various challenges and techniques involved in capturing the internals of graphics hardware. Then, we discuss in more detail how graphics state history can be stored, managed and presented to the user.

Previous works have not fully utilized and investigated historical information about OpenGL applications. We discuss how a graphics debugger can make this information easily available to assist in the OpenGL debugging process, addressing R1. In particular, we show how the use of historical information can be supported in a debugger's user interface, and motivate the features of GLDebug with specific use cases.

After completion of the GLDebug proof-of-concept prototype, a user study was conducted to evaluate the usefulness of the tool and address R2. This evaluation was fairly small in scale and scope, but seems to be the first of its kind: there is little or no research that attempts to evaluate the effectiveness of graphics debuggers.

Note that the results about the use of state history for debugging presented here are not only applicable to OpenGL. Our implementation is based solely on OpenGL, but other low-level graphics APIs such as DirectX are conceptually very similar. As a consequence, the contributions of this research can also be applied to other graphics APIs.

Section 2 summarizes the requirements of graphics debugging in general, and for using state history specifically. Section 3 gives an overview of related work. Section 4 introduces GLDebug and elaborates its design, including the user interface for making OpenGL state history easily accessible to developers. Section 5 details key areas of GLDebug's implementation. Section 6 explains some of the debugging use cases that can be addressed with GLDebug. Section 7 presents the results of the user study. Section 8 concludes the paper and points out some future work.

2 REQUIREMENTS

Common features of graphics debuggers include *state tracking*, *logging of graphics commands*, and the *inspection of buffers*. These features are widely used in modern graphics debuggers. In this project, we are also looking at novel features regarding the use of graphics state history, such as *logging of graphics states* and *comparison of graphics states*. In the following sections, we will describe all these features as requirements of graphics debuggers.

2.1 General Requirements of Graphics Debugging

State tracking is a functionality allowing a user to track, view, and potentially alter the state of the underlying graphics system. OpenGL is generally known to

be a state machine. How this machine is configured controls many aspects of how a command to the machine is processed. Bugs can easily be introduced by having the machine configured incorrectly [9].

As an example, consider a situation where a programmer is using a third-party library that makes changes to OpenGL state. Unfortunately, the programmer is not aware of these changes and thus subsequent OpenGL calls made by the program are not behaving as expected. But even if the programmer suspects this to be the cause, they still have to track down which part of the state is being altered.

In the above example, being able to inspect state is very helpful. The simple act of seeing what the state is and comparing that against what is expected saves the programmer from having to recompile code with debug instructions inserted to inspect state, or worse yet, from having to expend time learning that the bug is even related to OpenGL state. There are also instances of complex state interaction, where it is useful to be able to inspect several state variables at once. Presenting state information in a clear and easily navigable way facilitates this.

Command logging or **call logging** refers to a debugger logging commands being issued to the graphics API, and making the log visible to the user. This feature is useful as a reference, in a similar way to viewing OpenGL state: it helps verify that the *actual* behavior of the program is the same as the desired behavior. For example, this helps to make sure that a certain function is indeed being called, or that a certain argument to a function is correct.

Another useful, though rarer, aspect of this feature is being able to replay the commands that are logged. By doing this one can recreate a scene step by step, seeing the effect that each command has (visually and/or in the graphics state information). However, implementing this functionality is technically much more difficult than just logging calls.

Inspection of buffers is the ability of a debugger to query OpenGL for information contained in buffers belonging to the program being debugged, and then to expose this information in various ways to a user. Buffers can be used to store a variety of things, but the common inspection case is buffers storing texture (image) data. That said, support for inspecting other types of buffers exists in some debuggers, e.g. for buffers containing shader input data such as vertices. The way data is exposed can be visual or numeric, with different representations being appropriate depending on the buffer contents.

For example, a debugger could retrieve and allow inspection of the depth buffer, which helps determine if an object is being culled by the Z-test. Another use

case is the inspection of an off-screen texture that is being rendered to, a common technique in deferred shading/rendering [11]. Being able to visually inspect such a texture may be invaluable in seeing that the rendered image is as intended.

Shader debugging is functionality helping with shader bugs, which is becoming more and more important with the prevalence of shaders in modern graphics programming. Special support for shader debugging is necessary because of the shader pipeline being opaque: while input and output can be observed, what happens inside the pipeline is difficult or impossible to observe, making bugs that occur in the pipeline very difficult to diagnose and resolve. One of the popular shader debugging techniques is to instrument shaders so that additional information is output [14], allowing a programmer to read back the values of variables during shader execution – information that is normally inaccessible. Another technique is that of emulating the shader pipeline in software [13], allowing for much greater visibility and enabling identified requirements such as step-through debugging of shaders.

2.2 Requirements for Using State History

Our work here seeks to extend upon the ability of tracking the current state of an OpenGL program, by **tracking the state over the life** of such a program. This is similar to state tracking, with the additional requirement that captured information is persistent and is always available for recall. This contrasts with systems that only allow for viewing of the current state of a program, where previously captured information is not stored. Such concepts have been explored in the context of general purpose debuggers [10, 12], but have only been vaguely suggested for graphics debuggers [5].

In addition to tracking OpenGL state over the program execution, we also look at providing a means by which users can **compare the captured information** in a way that assists with debugging. It is important to report captured information to the users in a fashion that enables quick comparison of different states in order to facilitate the debugging process.

As an example of the above two requirements, a user should be able to record states from an OpenGL application that is running smoothly. When a bug is encountered, the user should be able to recall the state from when the program was running correctly, and compare that to the current, buggy state. The GUI should allow for a comparison such that the user is able to identify problematic states (if any).

3 RELATED WORK

3.1 Enabling Technology for Graphics Debugging

There are several technologies that enable and support graphics debugging, although they are not debuggers themselves. For example, there are systems available that aid in the capture of calls made to OpenGL, or that allow for querying of the state of OpenGL. A number of debuggers, including GLDebug, are built upon such systems.

Chromium [8] is a system for the manipulation of OpenGL command streams. Chromium uses a client-server model, with streams of commands being dispatched by clients to one or more servers from which the streams may be passed onto other servers. Each server can inspect and, if needed, modify the stream sent to it. Chromium can also be leveraged to manipulate the command streams, thus it is possible to alter the behavior of a program. These features are immediately useful in that they allow for both state tracking and command logging. However, Chromium is no longer being developed, leading to compatibility issues with recent versions of OpenGL.

BuGLE¹ is a toolkit designed to aid in the debugging of OpenGL applications. BuGLE makes use of filters that are used to intercept some or all OpenGL calls. Once a call is intercepted, it can be inspected, and modifications can also be made before the call is passed on to OpenGL. In contrast to Chromium, BuGLE is still being developed, so it has much better compatibility with more recent versions of OpenGL.

3.2 Graphics Debugging

The most actively developed graphics debuggers at present are commercial products, such as PIX² and Nsight³. There are also several academic projects in this area [7, 13], of which two major contributions are described below. However, little active research appears to be occurring in this area at the moment.

gDEBugger^{4,5} was one of the first commercial graphics debuggers to become widely available in 2004. It demonstrated many of the features seen in modern graphics debuggers, such as all of the general features discussed in Section 2. Furthermore, all these features were accessible through a GUI. The contribution of gDEBugger is in its pioneering of graphics debuggers in the commercial space, as well as offering many

¹ <http://sourceforge.net/projects/bugle/>

² <http://msdn.microsoft.com/en-us/library/ee663275%28v=vs.85%29.aspx>

³ <http://www.nvidia.com/object/nsight.html>

⁴ <http://developer.amd.com/tools/gDEBugger/Pages/default.aspx>

⁵ <http://www.gremedy.com/>

features incorporated with a GUI. gDEDebugger development has been discontinued as it became part of another debugger, CodeXL, which is discussed below.

Microsoft PIX is a commercial graphics debugger for use with DirectX on Windows as part of the Xbox 360 development kit, which is actively maintained by Microsoft. It is one of the few tools available for DirectX debugging. A notable feature of PIX is its ability to capture all of the commands used to create an image (a frame), and then replay these commands step by step on demand.

nVidia Nsight and **AMD CodeXL**⁶ are further examples of modern commercial debuggers. These tools provide many of the features mentioned in Section 2, including newer features for shader debugging, similar to those seen in GLSLDevil (see below). While they are available free of charge, their usage is limited to their developer's respective hardware.

There are tools that log calls made to graphics APIs such as OpenGL, e.g. **glintercept**⁷ and **apitrace**⁸. These tools log the API function calls made by an application to a file, and allow users to inspect this log, e.g. for profiling. Some of these tools (e.g. apitrace) also allow users to replay the log files and inspect the current graphics state during replay, similar to a graphics debugger.

GLSLDevil⁹ [14] is a tool specifically aimed at debugging the shader pipeline of OpenGL applications. GLSLDevil provides novel features in that it automatically instruments OpenGL shader code. The instrumented code then outputs extra information that can be used for debugging. GLSLDevil uses a GUI to present this information to users, showing the values of the variables used in a shader. It also supports some visualizations of those values, e.g. as images.

Apart from command logging and playback, historical information is not supported in any of the currently available debugging tools. A possible reason for this is that the storage and computation requirements make it non-trivial [10, 12]. Furthermore, the current research on graphics debugging exhibits a lack of evaluations of graphics debugging tools and their use in practice, which may make it an uncertain area to prioritize for development.

3.3 Debugging using History

The concept of recording the state of a program throughout its execution has been proposed for

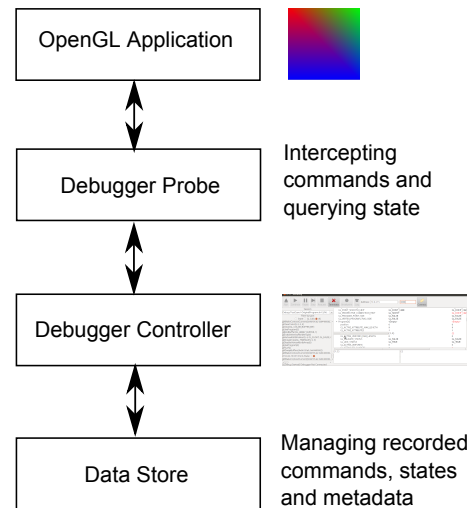


Figure 1: Architectural overview of GLDebug.

general-purpose debugging [10, 12]. The research in this area speculates that the ability to step back through a programs trace aids the user in certain debugging tasks. For example, such debuggers can help when a bug is found that is tied to a variable with an incorrect value. In this scenario, the debugger can be used to step backwards in time and find at what point the value deviated from appropriate values. Graphics debugging has some similarities to such general-purpose debugging scenarios: bugs often originate from some unintended state change [9], which is identified by inspecting the execution flow. However, the state machine aspect of graphics debugging is typically much stronger, with a reliance of outputs on a complex state and different types of potential bugs. Also, the technology involved in graphics debugging is different.

GQL (graphics query language) was created along with a debugging system by Duca et al. [5]. Similar to GLDebug, it enables tracking and logging the state and calls made by an OpenGL program over the course of execution. However, the historical information is only made available through an SQL-like language (GQL) that users have to learn, and there is no direct support for comparing states and highlighting of state differences. It is known that efficient use of a query language such as SQL depends strongly on individual ability and the user interface [4], hence it is questionable whether a textual query language such GQL can adequately support day-to-day graphics debugging tasks. GQL was not evaluated empirically to see if users find this approach effective or user friendly.

4 DESIGN

GLDebug is designed based on several high-level components, as shown in the architecture diagram in Figure 1. The *OpenGL application* is the program being debugged. It is executed on top of the *debugger probe*,

⁶ <http://developer.amd.com/tools/heterogeneous-computing/codexl/>

⁷ <http://code.google.com/p/glintercept/>

⁸ <http://apitrace.github.io/>

⁹ <http://cumbia.informatik.uni-stuttgart.de/glsdevil/>

which is a library that intercepts the OpenGL calls made by the application. Intercepting these calls makes it possible to capture information about the calls themselves as well as other data that can be inspected while the intercepting library has control.

The probe feeds the information it gathers into the debugger *controller*, which provides the GUI for controlling the debugging process. Through the probe, the controller can request graphics state information and influence the control flow of the application being debugged. The controller is also used to present information about the application to the user, and in particular let the user access the graphics state history in a convenient way. To support state history, the controller stores graphics states, OpenGL commands and related information in a *data store*. The data store is queried whenever historical information is needed. In the following paragraphs, the components of GLDebug are described in more detail.

4.1 Probe

The probe is the component responsible for capturing data from the program being debugged, and feeding that data to the controller. It is a shared library that provides the same interface as OpenGL. When a program is run, the probe is linked instead of the default OpenGL library. This means that all calls that would normally be made to the OpenGL library are passed to the probe instead. The probe allows for arbitrary code to be executed once a call is intercepted, hence taking over program control and allowing for both inspection and modification of OpenGL calls. It can process commands sent to it from the controller, such as for pausing the application, and send data to the controller, such as graphics state data that is queried by executing additional OpenGL commands.

There are several benefits of having the probe as a separate component of the system. For example, GLDebug can run on a computer separate from the computer running the OpenGL application. This provides benefits in terms of being able to run the probe and debug OpenGL applications on systems with less power and/or storage, such as mobile devices. Also, the probe can be developed independently of the other components. The downside is that there is additional work involved in developing a communication protocol for the probe and the controller.

The probe is lightweight, does not perform much processing and does not impede the OpenGL application. It is important that the probe does not alter the behavior of the OpenGL application. Similar designs can be found in other debuggers, such as the GQL debugger mentioned in Section 3, which has a separate process that processes the data captured from an application.

4.2 Controller

The controller is responsible for controlling the running OpenGL application and retrieving information about it through the probe. It is also responsible for storing the information in the data store, and making it accessible to the user through a GUI. Because of the distributed architecture of GLDebug, the controller and data store can be hosted on a more powerful system.

Figure 2 shows the controller GUI. The buttons at the top allow users to connect to a running probe and influence the control flow of the application being debugged, i.e. start, pause, stop and step through it. Furthermore, they allow users to set breakpoints on specific OpenGL functions, and request the graphics state from the application. The GUI also presents captured information back to the user. Graphics state information is presented in the right section of the window, below the top row of buttons. The table lists all OpenGL states variables with their values, and there is space at the bottom to show the value of a selected variable in more detail, i.e. in the case of longer state variables such as shader source code.

Note that the table on the right shows two graphics states, one in the left column and one in the right column. Differences in these two states are highlighted using color coding: unchanged variables are shown in black, variables with different values are highlighted in red, and if the values are the same but there has been a recorded state between the first and second state where the variables are not the same, then they are shown in purple. This allows users to quickly compare two graphics states. The states to compare are selected in the list on the left, which shows, among other information, the sequence of recorded states. The two columns of radio buttons are used to select the two states that are shown in the table on the right. The drop down list at the top lets users select different application sessions to view data from. So a user can select a state snapshot from one execution of a program, and compare it to another, or even compare state snapshots from two different programs.

Finally, the controller can show users a list of function calls that were captured by the probe. As shown in the radio button group near the top-left, the user may select to see only states, only function calls, or both together. This allows the user to explore the history of all captured states and function calls over the lifetime of the application.

Originally, the GUI had a multi-tab design where separate tabs were used to control the probe, and to view and compare captured information. However, this design was discarded in favor of the current single-window design after initial user feedback. Users found a multi-tab window to be too cumbersome as it required a lot of switching between tabs.

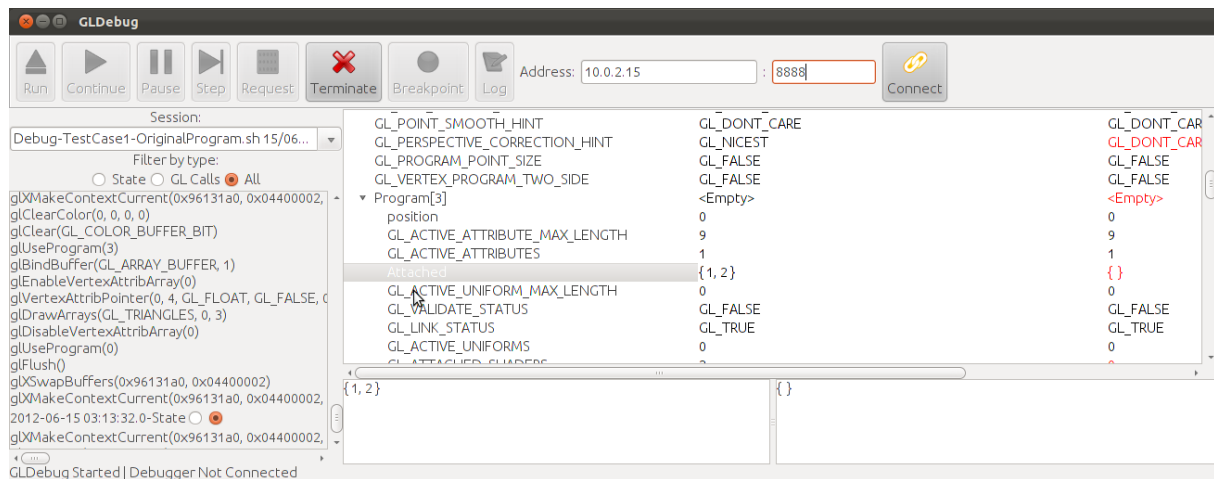


Figure 2: The GLDebug interface comparing two sets of captured state.

4.3 Data Store

The data store archives the OpenGL state information that is captured by the probe. This information consists mainly of state variables with names and values, with each graphics state containing hundreds of such variable-value pairs. Some variables are nested, i.e. they have child variables with values.

The data store also archives the function calls made by the OpenGL application that were logged by the probe. This includes the function name, parameter names and values, as well as the call order. Finally, the data store stores metadata about the logged information. This includes identifying information about the application being debugged and the debugging sessions, as well as timestamps for debugging sessions, states and function calls.

Our design makes use of a temporal database that performs delta encoding on stored information automatically. That is, when storing state information, it stores only the values of states that have actually changed. This means different states can be stored and recalled with minimal overhead, and comparison between the different states is somewhat simplified.

5 IMPLEMENTATION

GLDebug's probe was implemented using BuGLE (see Section 3.1) as a basis. The complexity and time requirements of implementing a debugger from scratch are significant. Using BuGLE as a basis greatly decreased the time required to develop the probe and implement the ability to capture OpenGL commands and state. However, BuGLE still had to be extended to meet the needs of GLDebug, e.g. with functionality for logging and sending information about OpenGL commands.

All communication between the probe and the controller is done through a single TCP connection. This

allows the probe to run on the same system as the controller or on another system, as required. The communication is primarily initiated by the controller, issuing requests to the probe, such as those for state, or those to start or stop the execution of the OpenGL application. When the probe receives a command, it attempts to carry out that command and reply to the controller as necessary. There are some cases where the communication is initiated by the probe, e.g. the sending of logged function calls.

The data store was implemented using a temporal triple store called PDStore, which was developed in our working group in a separate project. PDStore's ability to recall previous database states makes it possible to access any of the previously stored OpenGL states. Per default, the controller uses PDStore as an embedded database, so both run in the same process. As with many database systems, it is also possible to connect the controller to a remote PDStore database.

The controller was implemented using Java, while the probe had to be coded in a lower-level language (in this case C) in order to be compiled into a shared library. This separation was helped by the fact that both components communicate over a remote interface based on TCP, as explained earlier.

The implementation of the probe was fairly demanding, even when considering the use of BuGLE as a basis. It required extending BuGLE for capturing extra information and sending extra data, which required a detailed understanding of BuGLE's internals. Furthermore, a deeper understanding of linking was required in order to make sure that BuGLE was linked instead of OpenGL. For a more in-depth view of GLDebug's implementation see [16].

6 USE CASES

In the following we describe important use cases for the use of state history during graphics debugging. We de-

scribe what kind of bug is involved in a use case, its significance in real-world graphics applications, and how state history can help to find the bug more easily.

6.1 Incorrect Graphics State

GLDebug is useful in cases where bugs are caused by incorrect OpenGL state, and particularly in cases where the state shifts from intended to unintended values (so-called "snake in the grass" style bugs [10]). In cases where OpenGL is configured incorrectly, GLDebug makes it easier to view the values of state variables and thus find problems. However, GLDebug is of particular usefulness in the case where the state was configured correctly, and then shifts to an incorrect configuration. In such cases, being able to compare states can reveal not only the incorrect variable, but also shows in which state snapshot and at what time the problem occurred.

An example is a program that is rendering correctly, but then, through programming error, switches to an incorrect shader that results in a blank screen. In this scenario a comparison of the state captured when the program was performing correctly and incorrectly, respectively, would show that the shader source code is different. The user could then examine the source code from each of the different captured states, revealing that the incorrect code is being used in the error case.

Many bugs in OpenGL are caused by incorrect state [9], and being able to easily view OpenGL state is useful in diagnosing such bugs. These kinds of bugs can differ significantly in their severity depending on which and how many state variables are incorrectly set.

6.2 State Leakage

State leakage is a specific kind of incorrect configuration of state, where some code configures the state that then affects code elsewhere in a program. There are two major issues with this: first, the source of the issue is removed from the code where the problem occurs, making the bug hard to find. Secondly, the code causing the problem may not be available to the developer; for example, it may be part of a linked library. In addition to the above, while it may be apparent that a leak is happening, it is not always apparent which state variable(s) are being leaked and causing problems.

An example of this kind of bug is usage of an external library to draw a model using OpenGL. However, the library used has a bug in that it alters and does not reset the model-view matrix before returning control to the calling code. In this scenario, through no fault of the programmer using the library, bugs are introduced.

The use of external libraries is very common in graphics programming. There are numerous graphics libraries that build on OpenGL and other graphics APIs, and with

the continuing developments in processing power and computer graphics techniques, many of these libraries are subject to continuous change. Particularly for larger projects, it is rare that a single developer knows all the code in which graphics state is changed. As a result, state leakage problems can happen fairly easily.

GLDebug aids in these circumstances by making state information from different points in the application readily accessible, allowing users to find where and which state variables are being leaked. Being able to capture a state snapshot before and after the state leak allows users to use the state comparison features of GLDebug to identify the variables that have changed, and locate problematic state changes.

6.3 Missing Error Handling

OpenGL produces its own kind of errors, which require their own kind of error handling code. Without this error handling code, many errors would pass silently. An example is a compilation failure of a shader – something that would silently fail without error checking code, and simply lead to an incorrect output.

Programs often lack sufficient error handling code [17], and sometimes such code is omitted altogether. This can be particularly dangerous if errors happen silently and can lead to later problems, which is often the case with OpenGL. When these errors go undiagnosed, only to lead to problems later, finding the place where the error actually occurs can be particularly time consuming.

GLDebug simplifies catching of errors raised by OpenGL, meaning that such errors can be discovered even if a programmer has omitted error handling code. GLDebug can pause the execution of the program when OpenGL raises an error, and display information about the error to the user. By being able to catch such errors when they occur, GLDebug reduces their impact.

7 EVALUATION

A user study was performed to evaluate GLDebug and investigate in how far the use of graphics state history actually facilitates debugging. Interestingly, there do not seem to be any published studies on the usability of graphics debuggers at the moment. Our user study provides some insight into graphics debugging in general, and assesses the efficacy of the support for state history in GLDebug. It also serves as a building block for future studies in that area.

7.1 Methodology

In this evaluation participants were asked to complete graphics debugging tasks with and without GLDebug. By letting them use GLDebug for some tasks but not for others, all participants got an impression of how useful GLDebug can be. A mixed-methods approach was used to collect data during this study:

- **Think-aloud protocol:** While working on the debugging tasks, participants were encouraged to speak out their thoughts aloud and make comments at any point.
 - **Observations:** Participants were observed throughout the tasks, and significant observations were recorded.
 - **System Usability Scale (SUS):** After performing the debugging tasks, participants were asked to fill in the System Usability Scale [2] (a common usability questionnaire based on Likert-scales).
 - **Likert-scale questions:** Five custom Likert-scale questions were used for evaluating specific features of GLDebug.
 - **Open questions:** Open questions were used to ask what users liked and disliked about GLDebug, about improvements they could think of, and any other comments they may have.
1. Incorrect graphics state: An incorrectly configured Z-buffer, resulting in an output with a polygon that has clipping issues.
 2. State leakage: A call to an external library (for which the source code is not available) leaves texturing enabled, resulting textures being applied to polygons not intended to be textured.
 3. Missing error handling: A shader is not compatible with the shader model of the VM being used for the test, so that the shader is not being compiled and used, and the resulting polygon not colored correctly.
 4. Incorrect graphics state: An incorrectly configured model-view matrix that causes the output to be drawn progressively further and further away from the camera, instead of remaining static as desired.

Initially, also task completion times were recorded. However, this revealed one of the challenges when evaluating domain-specific tools for complex tasks, such as graphics debuggers: the performance of individual participants varied strongly, depending on how much graphics programming experience and programming skills they had, and other personal factors. This did not only introduce a lot of noise into the measurements, but also meant that some participants took an excessive amount of time to complete the tasks. Therefore, measurement of task completion times was abandoned after a few participants, and a maximum time of 15 minutes was allocated for each task. If a participant did not complete a task in the allocated time, the solution was presented and the participant could comment on it. To get meaningful results from quantitative measures such as task completion time, a lot of training would have to be incorporated into a study, or participants would have to be selected more carefully with regard to their graphics programming skills, to create a more homogeneous sample.

Each participant performed four debugging tasks: two with and two without GLDebug (i.e. using only text editor and compiler). Each task was performed by about half the participants with and the other half without GLDebug. The tasks were performed in the order presented below. The tasks were designed to each incorporate a single and unique bug. This helped us cast light on the utility of GLDebug for different kinds of bugs, and reduced any learning effects between the tasks that may have made tasks easier than usual. The tasks were modeled on real-world problems, but smaller in scale to allow for them to be solved in an appropriate time-frame. The four bugs involved were:

Before undertaking the tasks, participants were given general training on the use of GLDebug, as well as a briefing on each task, in the form of instructional videos. Participants were encouraged to give verbal feedback during the tasks, and following completion of a task. Following completion of all the tasks, participants were given the questionnaire to complete.

7.2 Results and Discussion

There were 7 participants, all of whom were male Computer Science postgraduate students. All but one had completed at least one course on Computer Graphics and had some experience in using OpenGL. Some had more extensive OpenGL project experience (more than a year of OpenGL development). The participants varied widely in both their general programming experience and their OpenGL programming experience. As discussed previously, this variation prevented us from using performance measures to assess the utility of GLDebug, but did provide us with the perspectives of users with different skill levels.

The results of the study were generally positive, indicating that participants found GLDebug useful for the tasks. Participants indicated that they found GLDebug especially useful when there were conspicuous state differences, or when they had a clear idea of what state variables to inspect. The less experienced users in particular were sometimes not sure which state variables were related to an issue, so they found it difficult to identify the relevant state changes. Users indicated in both the Likert-scale and open-ended questions that they liked the ability to compose a view of state over the course of program execution. However, users indicated they would like more flexibility and automation in how state was captured. In summary, people found GLDebug useful when it was clear how they could leverage

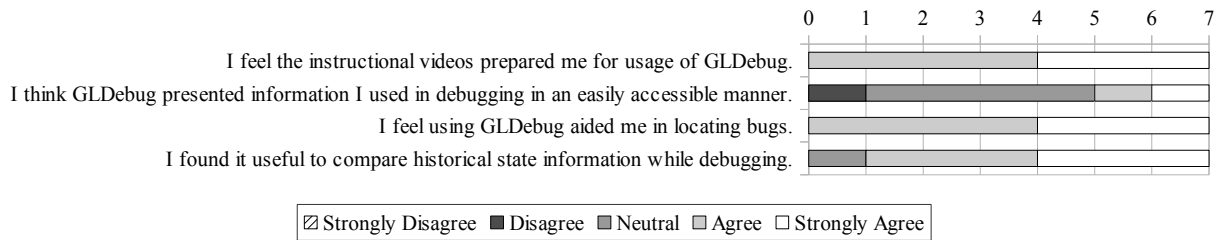


Figure 3: Participant responses to additional Likert-scale questions about the experience with GLDebug.

state information to debug a problem. Our results indicate that graphics state history can aid in debugging when users find information within that state that is clearly applicable to the problem at hand.

The average system usability scale score for GLDebug was 68.2, which is around average [1], and indicates that no serious usability issues are present. The participants suggested various improvements (see below), which helps to explain why the system got only an average score. For a research project such as GLDebug it is to be expected that it is not as polished and exhaustive in its functionality as a commercial product. The main point for this study was that debugging with state history was sufficiently supported.

Figure 3 shows the results of the additional Likert-scale questions. Q1 indicates that the instructional videos shown as training were perceived as sufficient. Q2 is in line with the results of the SUS, indicating no serious issue, but also indicating room for improvement in the presentation of information, which is discussed below. Q3 indicates that all users found GLDebug useful for debugging, which is a promising result for the prototype. Furthermore, Q4 shows that most users found the ability to compare captured state information useful.

The improvements suggested by the participants ranged from improvements to the GUI to thoughts on extra data that could be logged by the probe. Much of the feedback differed between the participants; for example, a common suggestion was making the GUI behave like an IDE the participant was familiar with. However, a strong majority of participants stated in the open question section that they wanted greater control over the ability to filter the information presented. Another desired feature mentioned in the open questions was the ability to automatically capture states based on certain conditions, such as each frame, or when a certain function call occurs. Filtering and conditional state capture would help to reduce the amount of information to that which is relevant for a specific bug. Participants also indicated a desire for functionality to show the original source code (if available) where an OpenGL call occurred, indicating the importance of putting the information provided by the debugger into proper context.

Our study has some limitations: a small sample size, a lack of professional graphics developers among the

sampled participants, and possible order effects. Small sample sizes are generally acceptable for qualitative usability studies, as experience shows that most usability problems can be identified even with few participants [6]. Furthermore, there is evidence that senior Computer Science postgraduate students as participants are a reasonable approximation of performing an experiment with software professionals [15]. Each task dealt with a different kind of bug to reduce learning between tasks, and training was given before undertaking the tasks to reduce the impact of learning. However, as all participants performed the tasks in the same order, it is possible that later tasks became easier. As the study was mostly qualitative, we do not consider this a severe problem. In conclusion, this study does provide evidence for the benefits of state history, but it should be validated with a larger sample taken from professional graphics programmers, or at least people with more extensive training and experience in graphics programming.

8 CONCLUSION

In this paper we investigated history-based graphics debugging – a practice that has remained largely unexplored in previous work. We illustrated how state history can be supported in a graphics debugger, and provided some empirical evidence for its utility. In summary, we have made the following contributions:

- The design and implementation of GLDebug, a graphics debugger with features for working with graphics state history.
- A discussion of use cases for history-based graphics debugging, and how they are supported by GLDebug.
- An evaluation of GLDebug, indicating that features for comparing historical states are useful.

Overall, historic state and call information seems to be useful for graphics debugging, and the evidence indicates that it would be a good idea to extend mainstream graphics debuggers with features similar to those of GLDebug. Our study also indicates that state history would be even more useful when combined with features for filtering it, to narrow down the flood of data

to relevant states. Another potential way to improve the use of state history is a better visualization of historical information. Filtering functionality and visualization of information are known to play a role for general-purpose debugging, so it would be interesting to investigate how they can further improve the use of graphics state history. Another area of interest is expanding the ability to specify when to capture states, such as capturing after particular OpenGL functions, or after drawing a particular entity.

9 ACKNOWLEDGMENTS

We would like to acknowledge the following people for their contributions to the GLDebug project: Bruce Merry, Meng-Da Lin, Osama Sagar, Heinrich Strauss, and Chen Xiliang.

REFERENCES

- [1] A. Bangor, P.T. Kortum, and J.T. Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human-Computer Interaction*, 24(6):574–594, 2008.
- [2] J. Brooke. SUS-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.
- [3] I. Buck, G. Humphreys, and P. Hanrahan. Tracking graphics state for networked rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 87–95. ACM, 2000.
- [4] Steven S Curl, Lorne Olfman, and John W Satzinger. An investigation of the roles of individual differences and user interface on database usability. *ACM SIGMIS Database*, 29(1):50–65, 1997.
- [5] N. Duca, K. Niski, J. Bilodeau, M. Bolitho, Y. Chen, and J. Cohen. A relational debugging engine for the graphics pipeline. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 453–463. ACM, 2005.
- [6] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383, 2003.
- [7] Q. Hou, K. Zhou, and B. Guo. Debugging gpu stream programs through automatic dataflow recording and visualization. In *ACM Transactions on Graphics (TOG)*, volume 28, page 153. ACM, 2009.
- [8] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, 2002.
- [9] Mark J Kilgard. Avoiding 19 common opengl pitfalls. In *Game Developer’s Conference, Proceedings*, 2000.
- [10] Bil Lewis. Debugging backwards in time. *CoRR*, cs.SE/0310016, 2003.
- [11] T. Möller, E. Haines, and N. Hoffman. *Real-time rendering*. AK Peters Ltd, 2008.
- [12] Guillaume Pothier, Éric Tanter, and José Piquer. Scalable omniscient debugging. In *ACM SIGPLAN Notices*, volume 42, pages 535–552. ACM, 2007.
- [13] Ahmad Sharif and Hsien-Hsin S Lee. Total recall: a debugging framework for gpus. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 13–20. Eurographics Association, 2008.
- [14] M. Strengert, T. Klein, and T. Ertl. A hardware-aware debugger for the OpenGL shading language. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 81–88. Eurographics Association, 2007.
- [15] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects - an empirical evaluation. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM ’08, pages 288–290. ACM, 2008.
- [16] Bryce Van Dyk. Using opengl state history for graphics debugging. Master’s thesis, The University of Auckland, New Zealand, 2012.
- [17] Westley Weimer and George C Necula. Finding and preventing run-time error handling mistakes. In *ACM SIGPLAN Notices*, volume 39, pages 419–431. ACM, 2004.

Virtual Reality Capabilities of Graphics Engines

Edward Peek

University of Auckland, NZ
epee004@aucklanduni.ac.nz

Burkhard Wünsche

University of Auckland, NZ
burkhard@cs.auckland.ac.nz

Christof Lutteroth

University of Auckland, NZ
lutteroth@cs.auckland.ac.nz

ABSTRACT

Desktop virtual reality has traditionally been the dominant display technology for consumer-level 3D computer graphics. Recently more sophisticated technologies such as stereoscopy and head-mounted displays have become more widely available. However, most 3D software is still only designed to support desktop VR, and must be modified to both technically support these displays and also to follow the best practises for their use. In this paper we evaluate modern 3D game/graphics engines and identify the degree to which they accommodate output to different types of affordable VR displays. We show that stereoscopy is widely supported, either natively or through existing adaptations. Other VR technologies such as head-mounted displays, head-coupled perspective (and consequentially fish-tank VR) are rarely natively supported. However, we identify and describe some methods, such as re-engineering, by which support for these display technologies can be added.

Keywords: virtual reality, graphics engine, head-coupled perspective, head-mounted display, stereoscopy

1 INTRODUCTION

A wide range of computer applications employ virtual reality (VR) concepts, including the general consumer applications that involve some sort of 3D virtual environment. Common examples of such applications are 3D modelling, computer aided design (CAD), video games, data visualisation, television and movies.

Recent commercial advances in consumer-level VR have lead to certain types of VR technology becoming cheap and of high enough quality to begin displacing the entrenched traditional technologies. Some examples of new devices that employ these novel VR technologies include haptic input methods such as Nintendo Wii Remote, Microsoft Kinect and Leap Motion Controller; head-mounted displays such as the Sony Personal 3D Viewer and Oculus Rift; and stereoscopic television sets, computer displays and projectors of which there are too many to name.

While attention and interest towards these technologies is slowly growing, support for them by VR applications is still limited. In the case of haptic inputs this is understandable since implementing natural user interfaces is a substantial departure from mouse/keyboard/controller based input systems. On the other hand, support for new VR display technologies is much less invasive and in some instances can even be achieved with no modification to the original software [10].

This work presents an investigation into modern software applications with the objective of determining what types of new VR display (not input) technologies are supported by these applications. We specifically look at graphics engines: reusable software components which handle output to VR displays and are shared by many applications. This allows a large number of applications to be covered with only the need to evaluate a few specific graphics engines. The following research questions embodies the objective of this study.

How far do modern graphics engines support consumer-level VR display technologies? How easily can support be added where they do not?

In answering these questions, we also make the following contributions.

- To provide a resource useful for determining which graphics engines are suitable for future application development and research in virtual reality.
- To identify common practises, shortcuts and interaction methods in engine design that makes them, in their current state, unsuitable for VR.
- To determine a general sense of how much attention is being paid to VR issues in consumer graphics engines.

In this paper we first give some background information about graphics engines and VR display technologies in Section 2, and describe some related work in Section 3. We then describe our methodology to evaluating the graphics engines in Section 4 and discuss our results in Section 5.

2 BACKGROUND

Graphics and Game Engines

A graphics engine is a reusable software component designed to render a 3D virtual environment. Graph-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WSCG'2013, June 24–27, 2013
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

ics engines can be distributed as standalone pieces of software or as part of larger systems, notably, but not limited to, game engines. This involves taking the current state of the simulated environment as input and rendering an image based on the lighting and shading model of the simulation. Real-time graphics engines are those that are capable of performing this process quickly enough to appear seamless to a user (typically around 30–60 rendered frames per second). real-time engines allow the simulation to be interactive and react to inputs from human users; a requirement of VR systems. In order to achieve real-time speeds, graphics engines normally delegate rendering to dedicated hardware and use algorithms and models that favour fast computation over physical accuracy.

VR Display Technologies

Virtual reality display technologies (also known as 3D displays) are the VR technologies that specifically deal with *visually presenting* a virtual environment to its user. These are used in addition to other VR technologies such as input systems and audio output, as well as the software that simulates the virtual environment. Within the context of this research, we do not consider the graphical rendering algorithms (such as rasterising polygons, lighting, shading and post-processing) to be part of a VR display technology, but rather part of the simulation logic. In this sense a VR display technology is only the hardware and software that *requests* graphical views from the environment simulation and *presents* them to the user.

Over time many different display technologies have been developed to satisfy this role. Nearly all of these operate on some variant of a camera metaphor; i.e. a virtual pinhole camera exists in the environment and regularly takes 2D snapshots which are then displayed on a physical display surface (such as a computer monitor). The components that make up such a display technology are the software that models the virtual camera, the hardware that displays images taken by the virtual camera, and the software interface that passes these images in the correct format to the display hardware.

There are several systems [4, 11] for classifying different VR display technologies based on different properties and generalisations. We utilise an alternative system that is based on software implementation requirements. In this paper we focus on consumer-level VR display technologies; specifically *desktop VR*, *stereoscopy*, *head-coupled perspective* and *head-mounted displays*.

The display properties most important to this study are how they are interfaced with from software, and how the rendering pipeline must be adapted to correctly reflect their perception model. What follows is a brief description of each of these display technologies, the

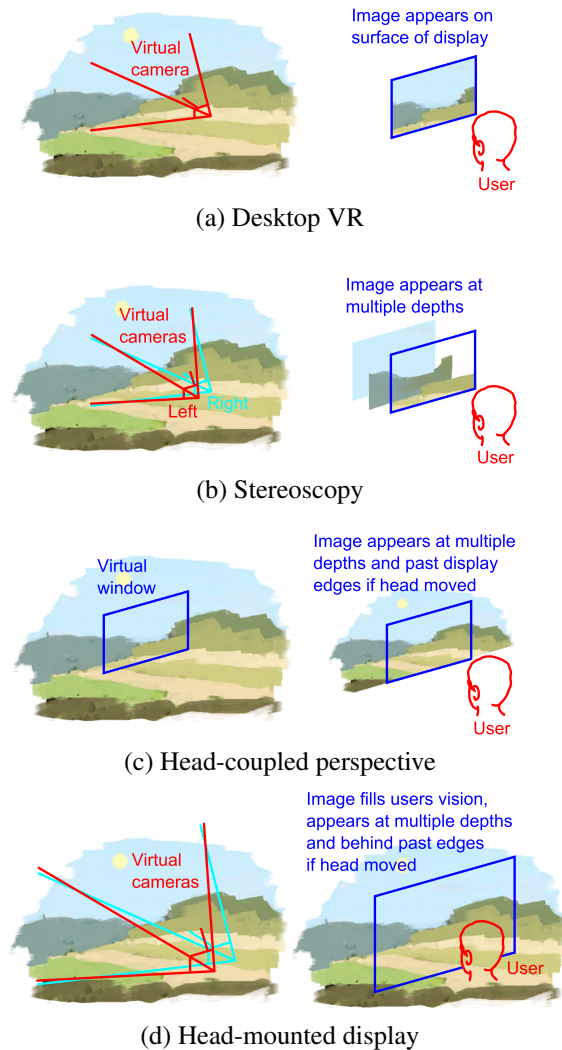


Figure 1: Depictions of differences between the VR display technologies in their simulation models and user's perception.

intent of which is to define the specific implementation requirements we use for this study.

Desktop VR has been the dominant form of presenting 3D virtual environments to their users since the advent of computer graphics. Desktop VR operates on a pinhole camera model, with a virtual camera controlled entirely by the simulation and a display capable of showing only a single image from this camera at a time. As the simplest form of VR it avoids many issues such as eye strain, increased computation cost and poor image quality that have hampered the use of more sophisticated technologies.

Because desktop VR is ubiquitously supported as the default output mode of virtually every graphics engine available today, we don't discuss it any further in this paper.

Stereoscopy is an extension of the desktop VR paradigm adapted for binocular vision. Stereoscopy achieves this by rendering the scene twice, once for each eye, then encoding and filtering the images in such a way that each image is seen by only one of the users' eyes. This filtering is most easily achieved through special eye glasses, the lenses of which are designed to selectively pass one of the two encodings produced by the matching display. Current methods of encoding are by colour spectrum, polarisation, temporally or spatially. These encoding methods are frequently categorised as *passive*, *active* or *autostereoscopic*. The difference between passive and active encoding is determined by whether or not the glasses are electrically active or not: passive encoding systems are therefore colour and polarisation while the only active encoding is temporal. Autostereoscopic displays are those that do not require glasses because they encode spatially, meaning that the physical distance between the eyes is sufficient to filter the images.

Consumer stereoscopic displays interface with computers in the same way as desktop VR displays (via video interfaces such as VGA or DVI). Since most of these interfaces do not have special modes for stereoscopy, the two stereo images are packed into a single image in a format recognised by the display hardware. Such *frame packing formats* include *interlaced*, *above-below*, *side-by-side*, *2D+depth* and *interleaved*.

Because these standardised interfaces are how the software passes rendered images to the display hardware, software applications are not required to know or adapt to the encoding system of the display hardware. Instead, all that is required for stereoscopy to be supported by a graphics engine is that it is able to render two images of the same simulation state from different virtual camera positions and combine them in a frame packing format supported by the display.

Head-coupled perspective (HCP) operates on a slightly different principle than desktop VR and stereoscopy. A virtual window is defined instead of a virtual camera, with the boundary of the virtual window mapped to the edges of the user's display. Thus, the image on the display depends on the relative position of the user's head, as objects from the virtual environment are projected onto the display in the direction of the user's eyes. This projection can be done using a off-axis version of the projection mathematics used in desktop VR.

In order to do this, the position of the users head relative to the display must be tracked accurately in real-time. Tracking systems that have been used for this purpose include armatures [19], electromagnetic/ultrasound trackers [18] and image-based tracking [12]. A limitation of HCP is that since the displayed image depends on the position of a user, any other users looking at the

same display will perceive a distorted image since they will not be viewing from the correct position.

Head-mounted displays are another type of single-user VR technology. HMDs combine the enhancements of stereoscopy with a large field-of-view and head-coupling similar to HCP. The perceptual model behind HMDs is to completely override the visual input to the users eyes and replace it with an encompassing view of the virtual environment. This is accomplished by mounting one or two small displays very close in front of the user's eyes with a lens system to allow for more natural focus. Since the displays are so close to the user's eyes, any part a display is only visible to one eye, making the system autostereoscopic.

An orientation tracker is also embedded in the head-gear, allowing for rotation of the user's head to be tracked. This allows the user to look around the virtual environment using natural head motion by binding the orientation of the virtual camera to the orientation of the user's head. This differs from HCP where it is the position, not orientation, that is tracked.

The software requirements to support HMDs are the same as stereoscopy, with the additional requirements that the orientation of the HMD must be considered by the graphics engine, as well as any distortion caused by the lens system to be corrected for.

In addition to these four technologies, there are numerous other types of VR displays that we do not address in this study. *Fish-tank VR* is not discussed because it is simply a combination of head-coupled perspective and stereoscopy. Furthermore, we do not consider more sophisticated VR technologies such as *multi-view displays*, *gaze-dependent depth of field*, *volume displays*, and *cave automatic virtual environments (CAVEs)* as they do not match our image of *consumer-level*. This is largely due to them being significantly more expensive (upwards of \$1000 USD), difficult to construct from off-the-shelf components or impractical to set up in many environments (CAVEs are an example of this).

3 RELATED WORK

General purpose graphics/game engines and virtual reality research are intrinsically linked, sharing several common goals. Both are highly dependent on realistic real-time 3D graphics and simulations, and both aim to generate a high degree of immersion and engagement. Because of this game engines provide many features that make them useful tools in scientific VR research. Correspondingly, advances in VR research often end up in graphics engines when they prove to be useful enhancements.

Lewis and Jacobson [8] explore the use of game engines for scientific simulation. The networking, graph-

ical and 3D scene management capabilities of the engines are noted as factors that make them useful for the variety of sample research applications they have been used for. Two of the engines mentioned in this article — the id Tech engine and the Unreal Engine — are investigated in our research, albeit using more recent versions. The authors do note however that for applications that require more sophisticated forms of VR, the base capabilities of the engines in question are not sufficient.

A more recent report by Trenholme and Smith [16] specifically evaluated common game engines for first-person virtual environments, building upon the work of Lewis and Jacobson. This work provides generic descriptions of the advantages and disadvantages of 6 reasonably modern (1–2 major versions behind what is current now) game engines for use in simulating virtual environments. However, this comparison does not consider the engines from a VR standpoint, so it misses out on recent trends. In addition to this, the capabilities of game engines advance at an extremely rapid pace and comparisons between previous generation technologies are not accurate for the current state of the art.

Where the capabilities of an engine are not sufficient for it to be used as-is for VR applications, but close enough to make it desirable, adaptations can be made to the engine to allow for its use. Lugin et al. [9] describe how the Unreal Engine 3 (again included in our research) can be adapted to support rendering in a CAVE system and accept input from a 3D tracked wand held by the user. This adaption was implemented as C++ plug-ins to incorporate the different forms of head and wand tracking, split across 6 networked clients to render the different sides of the CAVE with NVIDIA 3D vision to provide stereoscopy. Similar adaptations have been made to other engines to support more sophisticated VR such as with the Unity Engine and CryENGINE.

As well as game engines contributing to VR research, benefits also flow in the opposite direction, I.E. some VR technologies originally used for research have now become available in game engines. Litwiller and LaViola [6] discuss the implications of one such technology (stereoscopy) for gaming. They find that while there is no actual or perceived performance difference of the users' game scores when using stereoscopic 3D, the users did express a preference towards using stereoscopy over desktop VR. Sko and Gardner [14] investigate different technologies through implementing various uses of head tracking in games, while Andersen et al. [1] combine stereoscopy and head-coupled perspective (called fish-tank VR) in a first-person shooter game.

Despite the wealth of research into implementing VR with game engines, there is little general information on how well game engines support VR. This may be a result of the very specialised nature of many VR research

projects, and the tendency to focus on a single graphics engine or VR technology. By contrast, we discuss how far several current graphics engines can go to support various VR display technologies.

4 METHODOLOGY

Given enough time and effort, any graphics engine can be made to support almost any VR display technology. Different methods are available to do this, with a different amount of intrusiveness needed depending on how the software is designed and constructed.

Because measuring the amount of effort required to implement VR in a graphics engine is a difficult and inexact task, we have instead determined the *level of support* each graphics engine has for each of the VR display technologies. Additionally, quality factors are considered where applicable, as well as several generic properties of the engines that influence the implementation of these technologies.

Level of Support

With the flexibility of modern graphics engines it is not particularly meaningful to note features (particularly VR support) as *supported* or *not-supported*, since almost any feature can be made supported with reasonable effort. The addition of such non-native features is either facilitated through extension mechanisms built into the engine itself, built into the platform the engine runs on, or by re-engineering either of these two components. Some of the most common extension mechanisms built into graphics engines are node graphs, scripting, plug-ins and source modification.

In addition to these built-in extension mechanisms, it is also possible to add or modify functionality via re-engineering. This is required when the built-in extension points do not provide enough flexibility to implement the desired functionality. Re-engineering involves modifying the behaviour of a program by overriding portions of a program's original code or by replacing linked code libraries with modified variants. This will be described in detail along with the other extension mechanisms at the end of this section.

Level of support is measured by determining which extension mechanisms can be used to implement a desired VR display technology. Extension mechanisms with negligible differences have been combined (such as scripting and plug-ins), with two additional levels introduced for no extension needed (native support) and no in-engine support possible (re-engineering). Extension mechanisms are ordered by the proportion of engine code relative to non-engine code that implements

the VR support. The resulting levels of support and their ordering follows.

5. Natively supported
4. Via in-engine graphical customisation (including node graphs)
3. Via in-engine coding (scripting or plug-ins)
2. Via engine source code modification
1. Via re-engineering

This helps to answer our major research question and gives a sense of *engine support* and *engine flexibility* where high values indicate good VR support or flexibility, and low levels indicate poor VR support and low flexibility. It is important to note that this ordering is not a measure of the effort required to implement VR, but rather a measure of how well the engine assists this task.

We only report the highest level of support attained, as subsequently lower levels are practically always supported as well. In addition to presenting the highest level of support for each VR technology, we also indicate where third parties have demonstrated working implementations of the technology.

A brief description of each level of support follows.

Native In engines that natively support a VR technology, the developers of the engine have intentionally written the rendering pipeline in such a way that minimal effort is required by the user to enable VR rendering. All that is required is to check an option in the developer tools or set a variable in the engine's scripting environment. In addition to easily enabling the technology, the engines are also designed to avoid common optimisations and shortcuts that are not noticeable with desktop VR displays, but become noticeable with more sophisticated technologies. A common example of this is rendering objects with correct occlusion but at an incorrect depth [5], which causes depth cue conflicts under stereoscopy.

Graphical customisation Some engines are designed in such a way that the rendering process can be altered using custom tools with a graphical interface. One approach to this is via node graphs, where different components of the rendering pipeline can be rearranged, modified and reconnected in multiple configurations. Depending on what types of nodes are supported, it is sometimes possible to configure the nodes in such a way as to produce the effect of certain VR technologies. An example is shown in Figure 2, which depicts the Unreal Engine's material editing interface configured to render red-cyan anaglyph stereo as a post-processing effect.

Engine coding Practically every engine can be extended with custom code, using well-defined, but restricted, extension points. The two common forms of this are scripting, where the engine runs small programs/scripts in a restricted environment, and plug-ins, where the engine loads and runs externally compiled code. Both forms have access to a subset of engine features; however, plug-ins also have access to external APIs while scripts do not. Since this is the mechanism through which application-specific functionality is normally implemented, the engine features available to the custom code may be targeted more towards artificial intelligence, game logic and event sequencing, rather than controlling the exact rendering process.

Engine source code modification In addition to free open-source engines, some commercial engines make their complete source code available to users with the appropriate licence agreement. With access to the full source code any VR technology can be implemented, although the amount of modification required could be significant.

Re-engineering For engines that do not provide any of the above entry points for customisation, some amount of change is still possible through re-engineering. Re-engineering is a form of reverse-engineering where in addition to learning some of the workings of the program, some of its functionality is modified as well. The effort needed to fully reverse-engineer a rendering pipeline can be significant, so more minimally invasive forms of re-engineering are preferable. One of these approaches is function hooking, which is where the invocation of an internal or library function is intercepted and replaced with custom behaviour. Since a very large fraction of real-time graphics engines use the OpenGL or Direct3D libraries for hardware graphics acceleration, these libraries make reliable entry points for implementing visual-only VR technologies through function hooking. This approach has proved to be effective for adding stereoscopy to 3D games [10, 17]. We have also shown that it is also possible to implement head-coupled perspective in this manner [?], by hooking the OpenGL functions that load projection matrices (`glFrustum` and `glLoadMatrix`) and replacing the fixed-perspective matrices provided by the original program with head-coupled matrices.

Display Technology Support Criteria

For an engine to be labelled as supporting a specific VR display technology group, it must be able to satisfy the technical requirements of at least one actual display technology in that group (e.g. support for anaglyph stereoscopy indicates general stereoscopy support). Support can be achieved at any of the levels described previously, in which case all the technical requirements of the display technology must be implemented at that

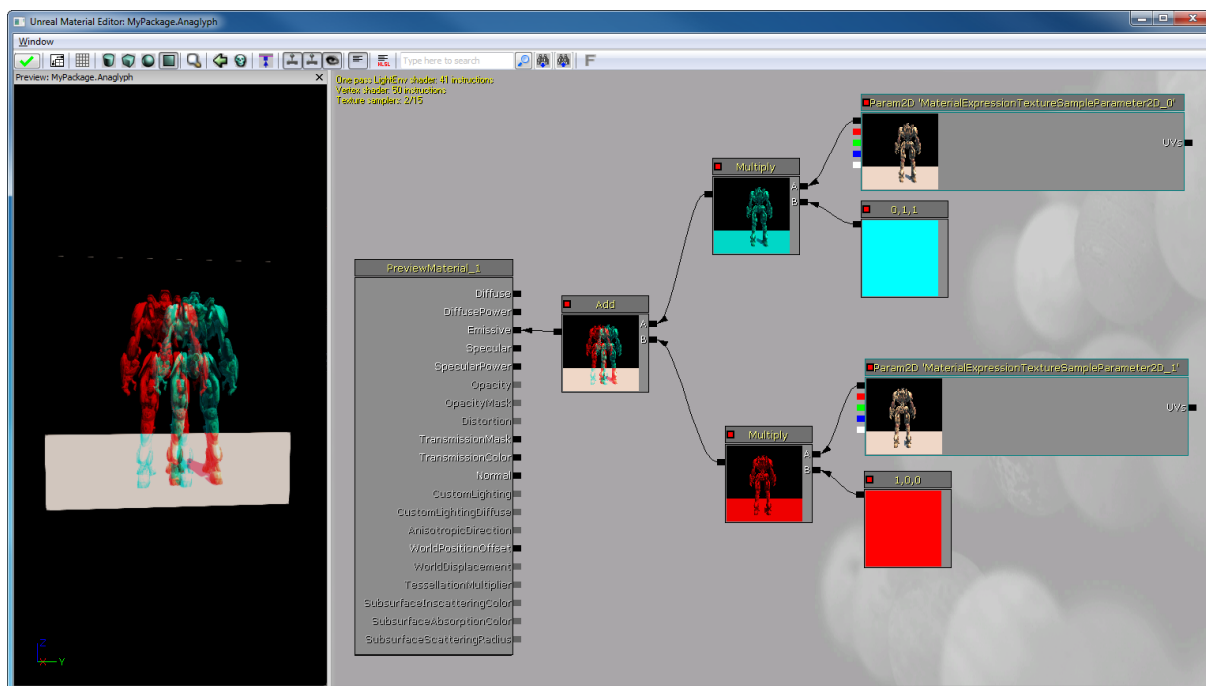


Figure 2: Configuration of the Unreal Engine to support red-cyan anaglyph stereoscopy, using the Material Editor. Adapted from [3]. Other stereo encodings can be supported in this manner, E.G. by interlacing the images for polarised stereo displays.

level or higher. The technical requirements of each display technology are the same as those outlined in Section 2.

VR Quality Factors

In addition to the technical challenge of implementing the VR display technologies just discussed, there are many secondary quality factors that affect a user's perceived quality of the VR experience. These factors arise because the implementations of the display technologies can not perfectly replicate the physical phenomenon they model. Since the differences are usually subtle, the user is frequently not consciously aware of them, but may instead experience some amount of eye strain, headaches or nausea. There can also be many different ways to implement any particular display technology, each of which balances different quality factors with other factors such as implementation cost. A prime example of this is stereoscopy, where at least ten different mechanisms to split images between the eyes have been used recently.

While quality factors are most inherently linked to the display hardware, appropriate software design can mitigate these issues, while careless design can introduce new issues. Because this study deals with the software implementation of VR display technologies, these software issues are of interest to us.

Examples of hardware quality factors that can be mitigated through software are crosstalk (stereoscopy), A/C breakdown (stereoscopy) and tracking latency

(HCP and HMDs). Since these factors are well established for their respective display technologies, there are well-known techniques to minimise issues they cause. The solutions are respectively reducing scene contrast, reducing parallax and minimising rendering delays. In most cases the engines this paper evaluates have non-native support for the display technologies associated with these quality factors, and subsequently do not follow these practices.

Incorrect software implementations can also influence the quality of the VR effect, which can occur due to carelessness, or as a result of optimisation for desktop VR. An example of this is special layers (such as the sky, shadows and first person player's body) at arbitrary depths in different passes. While this produces correct occlusion in desktop VR, the addition of the binocular parallax cue under stereoscopy reveals the incorrect depth, and creates a conflict between these two depth cues. This is not an uncommon issue due to the dominant nature of desktop VR, and serves as another example of where a naive third party implementation may not be as good as native VR support.

From these points it should be noted that while non-native VR implementations might meet the necessary technical requirements, other factors must be taken into account as well. Where possible we have pointed out these quality issues, but due to their dependence on a specific implementation and application it is difficult to make generalisations for a single graphics engine.

General Engine Properties

In addition to VR capabilities, this paper also outlines several general properties of graphics engines. These properties are chosen to assist researchers and developers in the selection of engines, to help identify trends of VR support and to classify the engines. What follows is a list of the general properties we considered useful. We do not elaborate on these properties as, being so general, they are largely self-descriptive.

- Developer interface
- Licences
- Programming languages
- Target platforms
- Version evaluated

Graphics Engines

The graphics engines of interest to us are those that are currently being used to render real-time 3D environments for research, commercial and other applications, and will likely continue to be used in the near future. We selected a representative sample of the most popular engines for this evaluation. The total number of graphics engines is greatly inflated by the number of graphics engines that are custom built for a select few applications. A secondary limiting factor is access to engines, as many are not made available to 3rd-party developers, only made available to established companies, or have prohibitively high licencing costs (in the order of \$100k+ USD). This has effectively restricted our investigation to graphics engines that are open-source or have free versions available with restricted access. Fortunately many normally expensive engines provide such versions, and so we are still able to cover a good range.

In addition to these restrictions, investigation of specific engines that are available to us have been prioritised according to the following factors.

- Engines should be in active development.
- An engine should have good community support, and be used in several applications.
- An engine should additionally have been considered in previous VR research.
- Engines designed for gaming should also have been used in non-gaming applications.
- Engines should focus on realistic and immersive graphics, and cutting edge technology.

The engines we evaluated can be put into 4 groups based on their licencing model, which also serves as a reasonably good overview of the general types of engine available.

Premium commercial engines (CryENGINE and Unreal Engine) are the most expensive and have the most comprehensive set of features. These are targeted towards large development studios that can afford the very high licencing costs to use the engine. These engines provide graphical tools to allow artists and game designers to use, while also allowing modification and extension of their source to implement application-specific behaviour. A recent trend has been for free versions of these engines to be released with specific restrictions, notably no source-code access and for non-commercial use only.

Commercial engines (Unity) are similar to premium engines but at significantly lower costs. They typically have slightly smaller feature sets or be intentionally simple and lightweight. Their main target audience is smaller (particularly indie) studios, individuals and hobbyists. Like premium engines, they typically provide graphical development interfaces to allow non-technical users to use them.

Previously commercial engines (Torque3D) are commercial engines that have at some point been made open-source. Reasons for this might be because newer versions of the same engine are now sold commercially, alternative revenue sources are being followed, because the engine is no longer competitive or to attract a larger user-base.

Open-source (OGRE and Irrlicht) are engines that are available for free under open-source licencing. They are frequently community developed, but sometimes also have backing by a commercial organisation. The quality and feature-sets of these engines varies dramatically, but usually falls short of commercial engines. These engines are typically fully code based, and do not provide graphical tools for development.

In addition to the engine categories included in this study, another major one is proprietary engines. These are those engines developed in-house for a specific application. None of these engines are included in this evaluation because they, by very nature, are not made available to third parties for development.

5 RESULTS AND DISCUSSION

The results of our evaluation can be found in Tables 1 and 2 with a discussion to follow.

The most obvious result from this evaluation is that almost none of the graphics engines evaluated support

VR technology	Stereoscopy	Head-coupled perspective	Head-mounted display
CryENGINE	5: Native [10] Support for both dual rendering and retargeting. Supports both manual and GPU driver frame packing.	3: Coding Access to camera matrices through C++ interface. C++ sufficient to access any head tracking method.	3: Coding [2] Stereoscopy supported natively, orientation tracking can be accessed via C++ plug-in.
OGRE	3: Coding [7, 10] OGRE rendering can be fully controlled and customised via implementation of all three display technologies.	3: Coding the C++ interface, allowing	3: Coding [13]
UDK	4: Graphical customisation [3, 10] Dual camera rig can be created using Unreal Kismet and outputs packed using the material editor.	1: Re-engineering* No access to custom camera projection from engine so re-engineering is needed if your licence does not include source code access.	3: Coding Stereoscopy through custom implementation, head orientation can be obtained via a custom DLL and bound to camera via script.
Unity	3: Coding [15] Dual cameras can be created and control via script, images can be packed as post-processing filter.	3: Coding Scripting supports custom camera projection matrices. Tracked head position can be obtained via C++ plug-in.	3: Coding Stereoscopy through custom implementation, head orientation can be obtained via C++ plug-in.
Irrlicht	3: Coding [10] Irrlicht rendering can be fully controlled and customised via implementation of all three display technologies.	3: Coding the C++ interface, allowing	3: Coding
Torque3D	3: Coding [10] Multiple passes of rendering are supported. This can be used to create the dual views and pack them in a compatible format.	2: Source modification Scripting interface to camera does not support off-axis projections, camera projection generation must be modified in code.	3: Coding [20] Head orientation can be accessed from an external tracker over TCP. Camera orientation can be updated based on this via script.

Table 1: Graphics engines' levels of support for various VR display technologies. *depends on licence

Name and Version	Interface	Licence	Code language	Platforms
CryENGINE 3.4.4	GUI Framework	Free for non-commercial use, Licence required for commercial use or source code access	C++ Lua	PC Games console
OGRE 1.8.1	Library	Open-source (MIT)	C++ Material scripts	PC Smartphone
UDK 2013/02b	GUI	Free for non-commercial use, Licence required for commercial use or source code access	C++ UnrealScript	PC Games console Smartphone
Unity 4.0.1f2	GUI	Free limited version Flat fee pro version Source code access via special licence	C# JavaScript	PC Games console Smartphone
Irrlicht 1.8	Library	Open-source (zlib)	C++	PC
Torque3D 2.0	GUI, Framework	Open-source (MIT)	TorqueScript, C++	PC

Table 2: General properties of graphics engines

a non-traditional VR display technology. The only engine that does is the CryENGINE, which natively supports stereoscopy in most of the formats used by modern stereoscopic displays. There are two explanations for this deficit. Firstly, that the developers of the engines do not believe these display technologies warrant the extra effort needed to support them. Or secondly, that they believe that the 3rd party support is good enough that native support is not necessary. It is our belief that the second point is the more likely, since all engines support stereoscopy through several 3rd party programs including NVIDIA 3D Vision.

In terms of how well the engines are designed to accommodate 3rd party VR support, most rate very highly with all but two instances having levels of support at *level 3: coding* or better. The two instances of lower support occurred when the scripting system did not provide enough control over the camera parameters. It is unknown whether the lack of access is intentional because the underlying rendering systems do not support arbitrary camera properties, or whether they were seen as unnecessary, not useful or just not thought considered.

In some cases the engine extension mechanisms do not have enough functionality to host the entire VR technology, but do provide communication functionality so that part of the technology can be offloaded to a separate process. This occurs when the scripting interface can't access the HMD or HCP head tracking values directly, but can indirectly over local TCP or UDP. Native code (e.g. C and C++) is normally needed to access the head tracking hardware. An example of this is Torque3D which does not provide any access to native code at levels of support above *level 2: source code modification*.

Of the three display technologies considered, HCP is the only for which we could not find any examples of 3rd-party implementations. Potential explanations might be that this is a less well-known technique, that it is a predominantly software technique and so is less easily commercialised, or more likely because it does not provide as good an effect as the other VR technologies.

The core point to take away from this work is that while the majority of graphics engines *do not* support most VR display technologies natively, they *almost always* provide enough flexibility such that support can be manually added.

6 CONCLUSIONS

We have described the mechanisms by which modern graphics and game engines may be extended to support non-traditional display technologies, particularly stereoscopy, head-coupled perspective and head-mounted displays. Where these engines do not have built-in extension mechanisms, or the ones that are provided are

too limited, these display technologies can always be implemented through re-engineering the engine.

Most of the engines evaluated do not provide native support for any non-traditional display technologies, and stereoscopy is the only technology that has any amount of native support in current versions of these engines. However several engines have support for head-mounted displays planned for future versions.

In the many instances where an engine does not provide native support for a display technology, support can usually be attained by developing a script or plug-in to produce the effect. Often this has been proved possible by other researchers or developers, and in many cases the source for the implementation is publicly available.

7 FUTURE WORK

As previously discussed, we believe the reason that most engines do not support most of the VR technologies evaluated is that there are still too few commercial displays that use them. As more exemplar displays become available this should start to change, and this can already be seen with several game engine developers (Torque3D, UDK and Unity) announcing support for HMDs (specifically the Oculus Rift) in future versions. It will be interesting to see whether support for specific technologies such as this will bleed through to other technologies as VR sophistication becomes a more important feature.

We have also considered a very small subset of the available classes of VR display technologies. Extending this evaluation to other technologies such as CAVEs, volumetric displays, multi-view displays and gaze-dependent field of view will increase the number of applications that benefit and also expose how engines can be adapted to cope with technologies substantially different from desktop VR.

In a similar vein, we have only evaluated 6 graphics engines which represents a tiny fraction of the entire population. Our preference towards selecting high speed real-time engines that have already been used for VR applications also means we did not consider any graphics engines used for applications such as CAD or scientific visualisation, which often have pseudo-real-time engines (in the sense that they react reasonably quickly to input, but not seamlessly).

We have also only considered the display side of VR, and ignored input technologies. While in many cases this can be done with little consequence, dependencies between the two have been known to cause problems. For instance mouse pointing depends on the virtual cameras projection properties which breaks down when there are multiple projections, as with stereoscopy, or the projection changes continuously, as with the tracking from HCP and HMDs. More work is needed to determine ways in which such input systems can be

accommodated for when using these display technologies.

REFERENCES

- [1] A.S. Andersen, J. Holst, and S.E. Vestergaard. The implementation of fish tank virtual reality in games.
- [2] Nathan Andrews. Crysis vr - head and gun tracking mod for the oculus rift, February 2013. URL <http://www.youtube.com/watch?v=TJx21yuCi7E>.
- [3] Christopher Berry. How to make a stereoscopic camera rig for udk, July 2011. URL <http://www.thebeardedberry.com/How%20To%20Make%20A%20Stereoscopic%20Camera%20Rig%20for%20UDK.pdf>.
- [4] Barry Blundell. On exemplar 3d display technologies. Technical report, Auckland University of Technology, 02 2012. URL <http://www.barrygblundell.com/upload/BBlundellWhitePaper.pdf>.
- [5] Slava Gostrenko. 3d stereoscopic game development - how to make your game look like beowulf 3d. In *NVIDIA Presentations at Game Developers Conference 2008*, San Francisco, USA, February 2008. URL <http://www.nvidia.com/object/gdc-2008.html>.
- [6] Joseph J. LaViola, Jr. and Tad Litwiller. Evaluating the benefits of 3d stereo in modern video games. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2345–2354, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9.
- [7] Mathieu Le Ber. Stereoscapy manager for ogre, January 2012. URL <http://sourceforge.net/p/ogreaddons/code/2986/tree/trunk/stereoscapy/>.
- [8] Michael Lewis and Jeffrey Jacobson. Game engines. *Communications of the ACM*, 45(1):27, 2002.
- [9] Jean-Luc Lugin, Fred Charles, Marc Cavazza, Marc Le Renard, Jonathan Freeman, and Jane Lessiter. Caveudk: a vr game engine middleware. In *Proceedings of the 18th ACM symposium on Virtual reality software and technology*, VRST '12, pages 137–144, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1469-5.
- [10] NVIDIA Corporation. Nvidia 3d vision, March 2013. URL <http://www.nvidia.com/object/3d-vision-main.html>.
- [11] Waldir Pimenta and Luís Paulo Santos. A comprehensive taxonomy for three-dimensional displays. In *WSCG 2012 – 20th International Conference on Computer Graphics, Visualization and Computer Vision*, pages 139–146. Union Agency, 2012.
- [12] J. Rekimoto. A vision-based head tracker for fish tank virtual reality-vr without head gear. In *Virtual Reality Annual International Symposium, 1995. Proceedings.*, pages 94 –100, mar 1995. doi: 10.1109/VRAIS.1995.512484.
- [13] Brian Ries, Victoria Interrante, Michael Kaeding, and Lee Anderson. The effect of self-embodiment on distance perception in immersive virtual environments. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, VRST '08, pages 167–170, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-951-7.
- [14] T. Sko and H. Gardner. Head tracking in first-person games: Interaction using a web-camera. *Human-Computer Interaction-INTERACT 2009*, pages 342–355, 2009.
- [15] Stereoskopix. Stereoskopix fov2go, January 2013. URL <https://www.assetstore.unity3d.com/#/content/2927>.
- [16] David Trenholme and ShamusP. Smith. Computer game engines for developing first-person virtual environments. *Virtual Reality*, 12:181–187, 2008. ISSN 1359-4338. doi: 10.1007/s10055-008-0092-z. URL <http://dx.doi.org/10.1007/s10055-008-0092-z>.
- [17] TriDef. Tridef 3d, March 2013. URL <http://www.tridef.com/products/pc>.
- [18] Colin Ware and Glenn Franck. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Trans. Graph.*, 15:121–140, April 1996. ISSN 0730-0301. doi: <http://doi.acm.org/10.1145/234972.234975>. URL <http://doi.acm.org/10.1145/234972.234975>.
- [19] Colin Ware, Kevin Arthur, and Kellogg S. Booth. Fish tank virtual reality. In *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, CHI '93, pages 37–42, New York, NY, USA, 1993. ACM. ISBN 0-89791-575-5.
- [20] David Wyand. Torque 3d and oculus rift, March 2013. URL <http://www.garagegames.com/community/blogs/view/22225>.

Physics-based Water Interaction and Shading: The SiViFlow Algorithm

David Sena
INESC-ID
Rua Alves Redol 9,
1000-029 Lisboa
davidsema@ist.utl.pt

Joao Pereira
INESC-ID
Rua Alves Redol 9,
1000-029 Lisboa
jap@inesc-id.pt

Vasco Costa
INESC-ID
Rua Alves Redol 9,
1000-029 Lisboa
vasco.costa@ist.utl.pt

ABSTRACT

Current real-time applications feature rivers that are pre-calculated off-line and present static animations and behaviours. These pre-calculated approaches have several limitations when used in real-time applications such as video games as they usually do not react to changes performed by the user. Due to the continuous pursue for better realism, the techniques used to simulate rivers have not only to improve the appearance of rivers but also allow them to adapt to dynamic changes performed in real-time. The approach presented in this work allows the dynamic generation of the river given any riverbed. The algorithm is also flexible enough to adapt the river flow in real-time. This approach not only accelerates the creation of realistic rivers but also increases the realism as the river is able to react to dynamic objects that come in contact with the flow, by properly adjusting its course.

Keywords

Water, Real-Time, River Animation, Flow Simulation.

1 INTRODUCTION

With the introduction of faster hardware and increasing demand for more realistic nature effects, researchers have been trying to create feasible nature models that are computationally viable and meet the constraints imposed by real-time applications. Nowadays applications such as video games try to simulate fully featured worlds with weather effects, large rivers and oceans, realistic animation systems among many other traits common in the real world. Due to the tight restrictions of real-time applications, an approach to simulate this type of phenomena would have to contain only the minimum amount of physical features necessary to make a river behave correctly and still leave enough computational resources available to draw a convincing visual representation of the fluid being simulated. The objective of the presented work is to create a new approach that simulates watercourses with any width, that flow correctly and are dynamic enough to be able to adapt to the features of their surroundings. A visually appealing representation of the flow being simulated is also included in order to be able to recreate with fidelity the watercourses from a visual standpoint. Our focus will reside mainly on the architecture description of the algorithm and less on implementation details or specific optimization issues. In order to focus the objectives of our work inside a broad subject such as fluid dynamics and as this work will be used in the context of video games, we decided to use real-time rendering techniques that allow the use of this approach in highly complex scenes. The final result had to be easily configurable both in

terms of visual appearance and physical parameters in order to allow this approach to be used in any setting. This would allow not only to change the visual features but also the behaviour of the river according to its surrounding, making it more flexible to adapt to different surroundings (e.g. it should be flexible enough to able portray both a tropical or a sea like environments). Regarding the dynamic flow simulation two main contributions were done in our work. First the automatic generation of a velocity vector field given an arbitrary river surface mesh. Given the mesh as input, the algorithm analyses and generates enough data to be able to create a vector field that describes not only the direction of the flow but also its velocity at any point. Second once we've calculated the vector field, we'll generate a realistic and adaptive flow behaviour which allows us to portray any amount of turns in a given river network and even take into account changes performed to the river channel such as dynamic objects altering the flow. This contribution takes into account the fact that the river surface mesh might have any width, have a complex river shape and that all the flow information drawn on screen is updated accordingly.

2 RELATED WORK

2.1 Navier-Stokes equations

The basis of most fluid simulation models both in Computational Fluid Dynamics and Computer Graphics are the Navier-Stokes equations. These equations allow us to represent a fluid by its velocity field and a pressure

field, varying both in time. If both fields are known at the initial time then we can describe the state of the fluid over time using:

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho} \nabla p + \nu \nabla^2 u + f \quad (1)$$

$$\nabla \cdot u = 0 \quad (2)$$

where \cdot denotes a dot product between vectors, ∇ is the vector of spatial derivatives, u and p are the velocity and pressure fields of the fluid, ρ is the density and ν is the kinematic viscosity. f is a vector representing external forces. Equation 2 is called the continuity equation and means that fluids conserve mass[Sta99]. The right-hand side of the equation 1 consists of four parts:

- **Advection** : $-(u \cdot \nabla)u$ which represents the process by which a fluid's velocity transports itself and other quantities in the fluid. In most simulations this represents the force that the surrounding fluid particles exert on a particle and causes it to transport itself along the velocity field.
- **Pressure** : $-\frac{1}{\rho} \nabla p$ causes regions with a higher pressure to accelerate the molecules away from that area.
- **Diffusion** : $\nu \nabla^2 u$ represents the force caused by the viscosity of the fluid.
- **External forces** : f represents forces that act on the fluid like gravity.

2.2 Approaches to Fluids Simulation

Physically-based water simulation has been an active research field for the last 30 years. Several different approaches have been proposed but usually they can be grouped into smaller distinct categories. In Figure 1 a schematic[GH06] is shown where the main types of water simulation are depicted.

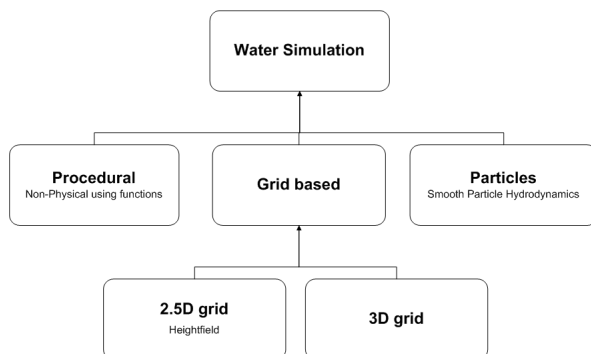


Figure 1: Water modeling techniques

The widest classification that can be made is a division between surface-based and volume-based techniques. The latter apply the Navier-Stokes equations to

model the liquid's physical flow properties. Amongst the volume-based techniques, we can find many different approaches. One of those categories is the Eulerian approach. This approach looks at fixed points in space, discretizing the domain in regular grids, either in 2D [Sta99][Fos96][WLL04] or 3D [Ngu07][CTG10]. Each grid cell stores both scalar quantities (such as pressure and temperature) and vector quantities such as velocity. In this approach the computational elements are fixed in space throughout the simulation and a finite difference method is used to solve the equations numerically. The major advantage of this method is the possibility to allow adaptive time steps and the inherent smooth liquid surface that it allows. On the other hand, this method suffers from a lengthy computational time and grid resolution limitations allied with aliasing in the boundary discretization. It also suffers from poor scalability in terms of computational power and memory consumption. Another approach is the Lagrangian, where the fluid is approximated by several discrete particles and their respective properties. Each point in the fluid is considered as a single particle, with a position x and a velocity u . In order to solve several problems regarding the discretization of the continuum using the Navier-Stokes equations, the method most commonly used are Smoothed Particles Hydrodynamics (SPH)[CBL⁺09][HKK07][DG96]. The approach taken by SPH is to define a smoothing kernel to interpolate physical properties (velocities, densities, etc) at an arbitrary position from the neighbouring particles, instead of defining each particle and their physical properties individually. This approach has two major drawbacks. First the smoothing kernel should be designed carefully because the stability, accuracy and speed of the SPH method largely depends on the choice of those kernels. Second there is quite a complex step in the Lagrangian method that is constructing a smooth surface for rendering. Many research works have presented possible solutions [vdLGS09] but up till now, the quality of liquid surfaces constructed from the whole bunch of particles is not as compelling as its Eulerian counterpart.

Among surface-based techniques, there are procedural methods which despite the fact that they don't model the whole fluid domain or some fluid quantities (e.g. pressure), usually represent the fluid in terms of velocity fields. These approaches don't start from the equations but pick a way to describe the state of the system (usually through a velocity field of the fluid), evaluating and updating it anywhere in space and time. Even nowadays this kind of approach is preferable because it provides an extremely simple approach to efficiently generate a fluid-like behaviour in a body. It also allows to control the animation of a body of water, something that is not as easy to obtain when using volume-based methods as in those approaches we would have to deal

with the discretization of partial differential equations, grids and solving systems of equations. Additionally most of previous methods rely on data that was computed with a fixed resolution, something that doesn't take into account a freedom of movement present inside most real-time applications and not present in movies or non-interactive demonstrations. One last advantage is the possibility to control several visual features of the fluid without having to recalculate the whole system, set the initial values and make sure that all the boundary conditions are well defined.

Method	Advantages	Disadvantages
Eulerian	Smooth Surface Adaptive time step	Memory usage Scalability Grid Resolution limitation
Lagrangian	More intuitive Irregular boundary	Smoothing Kernel Surface reconstruction
Procedural	Easy integration Extensible	Difficult to model some fluid values

Table 1: Water modelling techniques comparison

In Table 1 we show a summary of all the advantages and disadvantages of each technique.

For this work we chose the procedural approach because of the advantages described above and also due to the fact that it suits better the requirements of real-time applications.

2.3 Water Rendering

Fluids rendering is one of the most active fields inside Computer Graphics. As most of the physical behaviour of water couldn't be modelled at interactive frame rates inside real-time applications, developers and researchers focused most of their attention in getting as much visual fidelity as possible when rendering water. Reflection and refraction are elements that have been widely used in the simulation of water since the beginning of Computer Graphics [EMF02][GH06][PF05][Tes99]. Their use allows the user to see through the water and at the same time see the environment reflected on the water surface. This apparently trivial contribution fools the eye so much that most commercial products that include water algorithms sometimes only have these elements plus a wave generator. The most common way to describe reflection and refraction phenomena are the Fresnel equations [SJ09]. These equations allow us to describe the behaviour of light when moving between media with different refractive indices.

2.4 River Simulation and Rendering

A situation where fluid simulation is commonly applied to is when water flows between two or more bound-

aries, moving from a source into a sink. An example of that can be a river flowing where we have at least two river boundaries and the water flows to the river mouth or estuary. A river simulation can be decomposed in two main components: a simulation component where the physical behaviour is simulated and a visual component where the looks of the fluid are created. The work "Scalable Real-Time Animation of Rivers" [YN08][YNBH09] was able to simulate large scale rivers with realistic flow, yielding very appealing results. This work depicted a very realistic flow behaviour thanks to their new texture advection method, allowed real-time editing of the river channel with the respective flow adaptation to the new river boundaries and best of all it didn't depend on the scene complexity. Despite all these advantages there were still a couple of drawbacks. First the computational cost of the algorithm was linearly dependent with the projected river surface being rendered. Second the amount of data transferred between the Central Processing Unit (CPU) to the Graphics Processing Unit (GPU) is directly related with the Poisson-disk radius which increases linearly and quickly becomes prohibitive even with recent hardware. A final disadvantage was the need for the advection step to run on the CPU and the fact that this work assumed completely flat world profiles, excluding potential effects related with slopes of the terrain.

On the visual component there's a very visually appealing algorithm called Tiled Directional Flow [vH11]. This new algorithm offered several advantages over other flow simulation algorithms, was very cheap in terms of resources and yielded visually appealing results. They achieve a very realistic flow animation through the decomposition of the river surface in tiles, generating overlapping tiles all over the river channel (like a chess board on top of the river surface). Each tile has its own flow, local speed, direction and size of waves. By combining several normal maps together, the final result doesn't resemble sliding normal maps anymore and portrays a very pleasant appearance and animation. Even though the results of this algorithm were very satisfactory the fact that the authors have relied on the use of static flow maps limited the usage of this algorithm for big sized domains as it would require to either load a very large flow map or have some kind of spatial division algorithm to load the flow maps on the fly. Another disadvantage related with the use of static flow maps is that they can't take into account the influence of dynamic objects interacting with the river in real-time, which was something that had already been solved [YN08].

3 SIVIFLOW

SiViFlow is composed by two main elements: the Simulation Engine and the Visualization Engine. Figure 2

illustrates the block architecture of the SiViFlow algorithm. The Simulation Engine is where all the calculations related to physics of the river take place. This engine is divided in three main modules: the River Surface Generator, the River Particle Generator and the Flow Texture Mapper. From the programming point of view, the River Particle Generator and the Flow Texture Mapper make up a larger block called the River Particle Processor which will be described later in detail. The Visualization Engine is responsible for receiving the simulation data from the Simulation Engine and to output a graphical representation. This engine is divided in two main modules: the Flow Renderer and the Reflection.

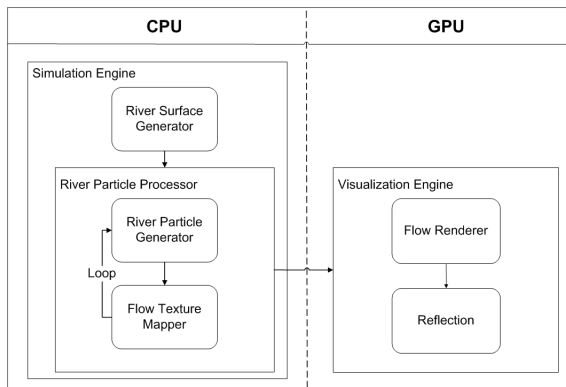


Figure 2: Block Architecture of SiViFlow

3.1 River Surface Generator

At this stage a river surface mesh needs to be created, which can either be done using an external modelling application or by generating it in real-time. At the beginning we don't know how many vertices go from one shore to the other in one single section of the river, so we start by calculating the river width and flag which vertices can be considered shore vertices. A river section is a set of vertices that are placed between two shore vertices and form a line that is perpendicular with both river shores as shown in Figure 3. In order to find out which vertices are shore vertices, we start by identifying the first vertex from the river mesh and calculate differences in distance between this vertex and all the other vertices that follow. When we reach the end of the river section we're processing, the difference stops increasing and it means we've reached the vertex which is on the same shore as our first vertex (the shore vertex right next to the one we're processing), thus the last vertex we processed belongs to the opposite shore. With that we calculate the river width (see Algorithm 1).

Algorithm 1 sums up all the steps taken during this pre-processing phase. The only input information required are the river mesh vertices. The algorithm starts looping from the first vertex which we know it's a shore vertex as it's located in a corner of the river mesh. We compare

```

for all vertices do
  if vertex is a shore vertex then
    Flag vertex
    RiverWidth(vertices)
    DistanceToMargins(vertices)
    CalculateFlow(vertices)
  
```

Algorithm 1: River surface generation algorithm

the width between this first vertex and the following vertices, making sure to always store a new width if the value is larger than what was previously stored. When the section of the river ends and we're processing the shore vertex which is on the same shore and right next to the first one, the distance between both vertices will be smaller than the full width of the river. We store the current width value and the amount of vertices that go from one shore to the other. At this stage we know the river width at each section as we have looped through all the river sections that compose the river surface. We also know the amount of vertices that go from one shore to another, allowing us to flag the vertices that belong to the river shore. These vertices need to be handled differently because they'll be used for calculating the flow. Now for each vertex in the river mesh, we store its distances to each of the river shore vertices at their river section. This information will later be used to calculate the flow velocity. Lastly we calculate the river flow at each river section, storing the information in every vertex. Both the flow velocity and flow generation will be described in more detail in the following sections.

3.1.1 Flow Generation

In order to calculate the flow we pick two shore vertices in the same river section, then we calculate their midpoint and translate in the positive up axis, as shown in Figure 3 where the up vector used is aligned with the y axis.

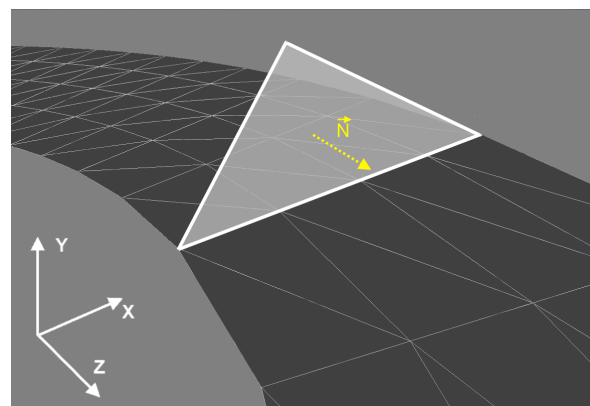


Figure 3: Flow vector created from a plane defined by two shore vertices and their midpoint translated in the +y axis

With these three points we generate a plane that will allow us to create a vector that is perpendicular with the river section being processed. As the flow is constant for each river section and is parallel to the margins, the normal vector of the plane describes correctly the flow direction of that section as shown in Figure 3. Since the generated plane has two possible normal vectors, the normal generation procedure must take into account this direction and return the correct normal vector. In the end we have a flow field that is as detailed as the mesh of the river surface and where each vertex contains its own flow vector stored.

One advantage of generating the flow this way regards its flexibility to dynamically recalculate the flow when an object interacts with the river. In case a dynamic object alters the course of the flow, the boundaries of the object will be used to recalculate the new flow and will substitute the shore vertices that were previously used.

As the values are tied to the river mesh and the collision vertices are known, SiViFlow is able to recompute the flow of the river and immediately reflect the changes.

3.1.2 Flow Velocity

In order to obtain the flow velocity we calculate a stream function field(Ψ) for the river channel flow using an existent interpolation scheme [YN08][YNBH09]. At this stage we have all the information required to calculate the following equations. We run for each vertex all the Equations 3, 4 and 5 [YN08][YNBH09] and store their values.

$$\Psi(P) = \frac{\sum_i w(d_i) \Psi_i}{\sum_i w(d_i)} \quad (3)$$

with P being the position of each river surface vertex, d_i the distance from point P to the each of the boundaries, Ψ_i the stream function value of a margin and the weighting factor w is:

$$w(d) = \begin{cases} d^{-p} \cdot f(1 - \frac{d}{s}), & \text{if } 0 < d \leq s, \\ 0, & \text{if } s < d, \end{cases} \quad (4)$$

where s is the radius used to search for boundaries, p is a positive real number and f is defined as:

$$f(t) = 6t^5 - 15t^4 + 10t^3 \quad (5)$$

3.2 River Particle Processor

River particles are a concept we created in order to sample information from our domain and retrieve its values. As we want to be able to handle large watercourses, it's not feasible to rely on loading all the river surface information to Video RAM (VRAM) every frame. In our

case we're interested in getting only the visible river mesh values so we can retrieve and send them to be rendered on the GPU. One of the main features of the river particles is that they're created in screen space in order to guarantee a uniform distribution of the particles over the visible domain at each frame. The reason for generating these points in screen space is that as each particle contains a defined radius to make sure no two particles are too close to each other, analysing this problem in screen space guarantees that these radius disks maintain a uniform radius. In world space these disks would be ellipses which would make the detection of overlapping particles harder. Another advantage of this scheme is that we only process visible information as we eliminate all non-visible particles which minimizes the waste of resources. There are some similar approaches to ours such as texture sprites [Ney03] and wave sprites [YN08][YNBH09].

3.2.1 River Particle Generator

We start by generating several randomly distributed points, generating a Poisson-disk pattern using a modified boundary sampling algorithm [Bri07][DH06]. We've adapted this algorithm to start from a fixed set of points instead of a random point. An advantage of this algorithm is that it guarantees that all points are equally distributed over the given domain, which in this case as we're aiming to generate particles in screen space, means they're all equally distributed over the screen.

In the end of running this algorithm, we end up with a set of points that we'll convert to river particles. In order to generate a 3D world position for each of these points (after being generated we only have their 2D coordinates) we proceed as Figure 4 shows. A ray is cast for each particle and we store the collision point between the ray and the 3D world. Using this method we can compute at each frame, for each point, its 3D world position. Besides calculating the world position we also calculate other features such as global identifiers to be able to identify each of the particles, velocity and flow. Unlike other algorithms [YN08][YNBH09], we don't advect our particles during our CPU update loop. The reason for this is due to the fact that our particles aren't concerned with the fluid's motion, they're simply a way to sample the necessary information in screen space and send it from the CPU to the GPU. An inherent advantage of not having to advect particles during the update loop is that it allows us to offload the work from the CPU to the GPU.

All of this information will allow us to find out in the next stage what's the nearest flow data to load into the flow texture. We just search inside a radius r for the closest vertex and assign that flow information to the

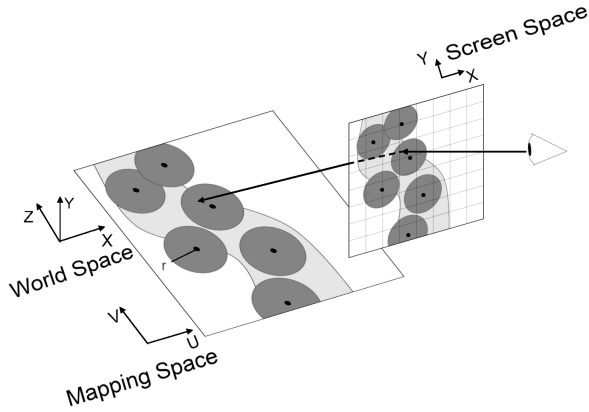


Figure 4: Ray cast performed from camera position and mapped into world space to obtain each particle's world position

river particle. The value of the radius r used for each situation was obtained through trial and error, although more sophisticated approaches can be used. This step differs from [YN08][YNBH09] as they first render the river surface to a buffer inside the GPU, find out which particles are inside the river surface and then query each individual pixel to find out which particle sits inside. Our approach despite being a bit more computationally intensive, doesn't have the inherent problems that might arise from relying in performing constant transfers between the CPU and GPU.

3.2.2 Flow Texture Mapper

In order to feed the GPU with the information required to render the flow, we used a flow texture and an auxiliary texture. Similar ideas have been explored by other authors [YN08][YNBH09][PF05] to achieve other objectives. In our approach we store all the information we need inside each color channel and read it back when it reaches the GPU. In Figure 5 we can see the distribution of each of the components in both the flow texture and auxiliary texture.

R8	G8	B8	A8	
Particle Index Number	Flow Vector x	Flow Vector y	Flow Vector z	Flow Texture
Velocity	Depth	Slope	-----	Auxiliary Texture

Figure 5: How each component is stored inside each of the 8 bit size texture channels

These textures will store the river particles previously generated using each of the color channels of the texture.

In the flow texture we will store for every entry data such as the global identifier of the river particle and its respective flow. The identifier in this texture will be used as a way to look-up the remaining data from the auxiliary texture. For each entry of the flow texture, we

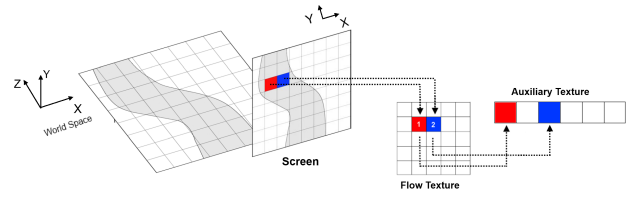


Figure 6: Storage scheme used in the flow and auxiliary textures

```

while true do
  for all particles do
    if Particle is outside of frustum then
      Delete Particle
    if Particle violates the minimum distance criterion in Screen Space then
      Delete Particle
    Insert new particles to keep the Poisson-disk
  for all new particles do
    Convert to river particles
    Write new data to the flow texture
    Write new data to the auxiliary texture
  Render

```

Algorithm 2: Application loop

store the flow information that covers that pixel. For performance reasons we used a flow texture that had a lower resolution than the screen resolution being used. The auxiliary texture will have other parameters such as velocity, river bed slope and river depth. In Figure 6 we can see how each river particle is stored in a smaller sized version of the flow texture and how the global identifier for each particle will be used to address the auxiliary texture.

In Algorithm 2 we can see that the whole update process is performed at every frame update. First we start by having to delete the particles that are not visible as they are wasting resources and won't affect the final result. Then we need to delete the particles that are too close to one another violating the initial Poisson-disk requirement that all particles must be no closer to each other more than a specified radius distance. In order to keep a reasonable number of particles in screen, after deleting all the unnecessary particles we generate new ones using the previously mentioned algorithm. After this, for all new particles, we have to convert them to river particles by calculating all their features. To end the algorithm we fill the flow and auxiliary textures with the current data from that frame and get them ready to be sent to the GPU.

3.3 Visualization Engine

The Visualization Engine is the last stage of SiViFlow and consists of mapping a material to the river surface

Access flow texture to find covering sprite index and flow information
 Access auxiliary texture to find velocity and depth
 Use flow information for Tiled Directional Flow algorithm
 Use new normal vector for reflection
 Blend all the elements

Algorithm 3: Fragment Shader of the Visualization Engine

mesh. This stage is divided in two main elements: the Flow Renderer and the Reflection algorithm which are implemented in a fragment shader. We start by accessing the flow texture and consult the river particle identifier of this pixel. In order to optimize the texture look-up, the flow information is also saved during this operation. Now we can use the river particle identifier to look-up the rest of the parameters contained inside the auxiliary texture.

We also use the flow information to generate the normal which will be used to compute the scene's reflection. All the steps of the algorithm are summed up in Algorithm 3.

3.3.1 Flow Renderer

Our flow algorithm is based in the "Tiled Directional Flow" described in [vH11]. In our approach one of the main differences is that all the flow information being fed to the algorithm is not based on a fixed flow map but comes from our flow and auxiliary textures. This allows us to work with a much smaller amount of information at each render cycle because our flow texture only contains information that's visible during that frame. The fact that our flow texture is updated every frame, means that we can change the flow if any dynamic object changes river flow.

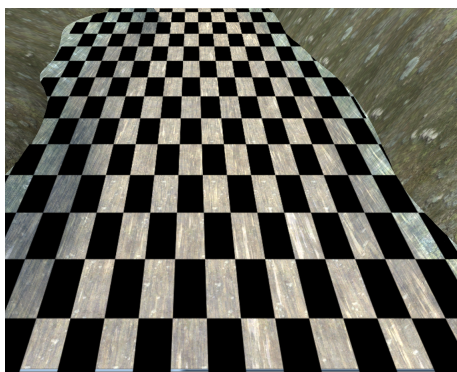


Figure 7: Example of the tiling division performed on top of the river surface for the flow algorithm

The way this approach works is by dividing a river channel in tiles, similar to a chess board. We show this division in Figure 7. Each tile is independent from its

peers and its composed by several normal maps. In order to get a more convincing look, we used for each tile four normal maps that are combined and blended together. First the regular normal map is loaded for the tile being processed. Then we sample a normal map with half a tile shift in the x direction and we rotate it in order to have independent features from the previous normal map. These two tiles are blended together using a blending factor. The next two normal maps follow the same idea, the first one is sampled with a shift in the y direction and the second is shifted in the x and y direction. Both normal maps are rotated and combined together using the same blending factor. To get the final normal value, both normal maps are blended once more by using the same blending factor. To conclude this final blending step of normal maps a scaling operation has to be performed. This scaling operation avoids the problem of having a resulting normal closer to the actual average normal, which is common when several normal vectors are added together.

3.3.2 Reflection

In order to simulate dynamic reflections of objects on our river surface we used the well-known planar reflections algorithm [AMHH08][Eng03][PF05].

This approach has been widely used since the introduction of the programmable pipelines because of its ease of use and how inexpensive it is in terms of resources. An example of this technique can be seen in Figure 8 where it is visible the reflection of the house near the shore. This technique is based on the use of a texture called a reflection map, which is an inverted version of what it is visible above the water level and that we want to reflect. To obtain a reflection map, we start by defining a clipping plane, which has to be about the same height as the river surface.

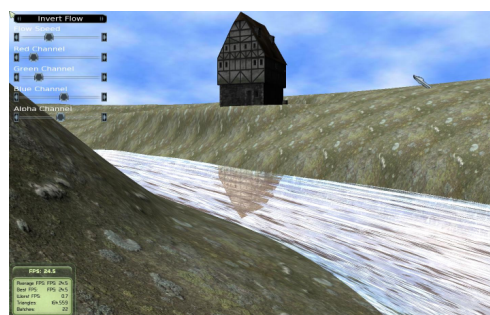


Figure 8: Example of the final scene appearance using planar reflections

This clipping plane will be useful to cut all the geometry below the river surface that we're not interested in rendering. If we didn't clip the contents below the river surface, we would reflect also the contents of the river which would break all illusion of reflection. After that we save an inverted copy of this clipped scene

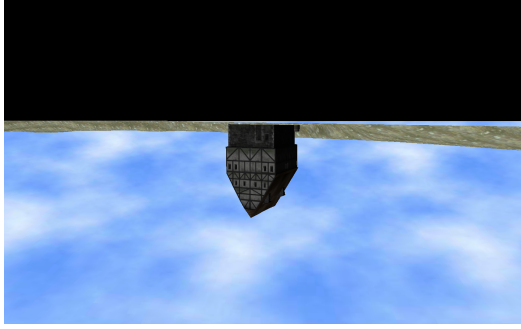


Figure 9: Example of a reflection map created clipping all the geometry below the river surface and reflecting the remaining contents

to a texture as in Figure 9 where we can see the contents of Figure 8 inverted and the whole river surface clipped. As the inverted copy is saved into a texture, we can send it to the GPU in order to be read inside our material. When we render our river material, we sample the correspondent pixel and blend the reflected information with the color we'll be outputting from the fragment shader.

4 RESULTS

This section provides the results and corresponding analysis for both the Simulation Engine and Visualization Engine. For the Simulation Engine we consider all the stages that deal with the creation, update and destruction of river particles and have to pack the required information in order to make it readable by the GPU. For the Visualization Engine we consider the Flow Renderer and Reflection stages which are comprised within the river material. We implemented our approach on top of the open-source game engine Ogre¹ version 1.7.3.(Cthuga). The algorithm was coded in C++ using the DirectX 9 API renderer provided by Ogre and the shaders were coded in HLSL. The platform used for testing is a computer with an Intel Core i7 running at 3 GHz with 8GB of RAM, a Nvidia GeForce GTX 480 with 1536 MB of VRAM and Microsoft Windows 7 x64 as the operating system. In order to measure the timings that each stage of our algorithm takes, we used Intel's VTune Amplifier² for the code that runs in the CPU and Intel's Graphics Performance Analyzer³ to profile the timings in the GPU.

4.1 Simulation Engine

The Simulation Engine is composed of the River Surface Generator, the River Particle Generator and the Flow Texture Mapper. As the River Surface Generator only runs once to create the river surface mesh

and it is not part of the application loop, all the measurements performed focused on the remaining components. This means that the application update loop can be divided in two main phases: the River Particle Generator and the Flow Texture Mapper. In Table 2 we can see how many particles were used in average to sample the whole screen.

Screen Resolution	Average Number of River Particles	Frames per second
800x600	336	32
1280x800	369	30
1440x900	407	29
1680x1050	384	28

Table 2: Average amount of river particles existent for different screen resolutions and average frames per second obtained throughout the tests.

We didn't use a fixed number of particles across all tests due to the nature of the sampling method we used. As the Poisson disk method randomly samples points across the domain, in order to minimize possible holes, some distributions might require more points than others. As shown when the screen resolution increases, the average frames per second decreases. This is due to the fact that as screen resolution increases, more particles are used and more pixels need to be processed in the CPU in order to map the best particle into the flow texture.

4.1.1 River Particle Generator

As mentioned in Section 3.2, the River Particle Generator is responsible for deleting river particles that are not visible, delete river particles that are too close to one another and generate new particles making sure they're converted to river particles.

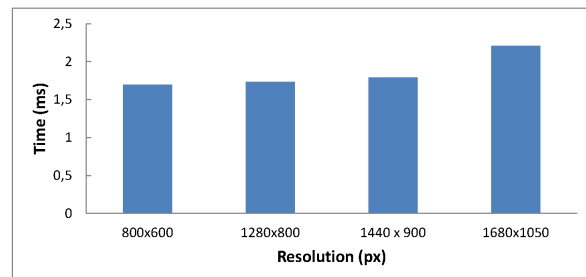


Figure 10: Time taken in milliseconds to update the river particles

In Figure 10 we can see that the time taken to update the river particles varies slightly across different resolutions. It's possible to see a slight increase in time taken to update the particles as the resolutions increase but the difference is less than 0.4 millisecond from the smaller resolution to the largest one.

¹ <http://www.ogre3d.org/>

² <http://software.intel.com/en-us/intel-vtune-amplifier-xe>

³ <http://software.intel.com/en-us/vcs/source/tools/intel-gpa>

4.1.2 Flow Texture Mapper

The loading of new data into the flow and auxiliary textures is a step that must run at every frame and is performed in the Flow Texture Mapper. We'll start by analysing the time taken by the flow texture and after we'll analyse the auxiliary texture. As soon as we started profiling the application, we saw that the loading of data into the flow texture was the step in the whole algorithm that consumed more time. We used for all tests a flow texture with 64 by 64 pixels, meaning we had to map the screen resolution being used to the size of the flow texture and find the best particle that cover that section of the screen.

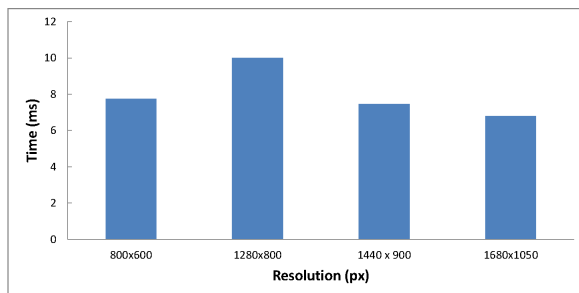


Figure 11: Time taken in milliseconds to load all the data into the Flow texture

We can see in Figure 11 that all the values tend to stay relatively close to one another. This is due to the fact that this step is not only our application's bottleneck but it's not directly influenced by the screen resolution as we always load a flow texture with the same dimensions. Upon closer look we noticed that the operations that were taking most of the time were finding the particle that better covers the largest amount of the pixels that are being processed and making sure that there were no sections of the texture without river particles. As the flow texture has a smaller size than our screen resolution, we map an amount of screen pixels that correspond to a single entry in the flow texture and process it. We retrieve all the river particles that cover this section and choose the one that covers the largest amount of the area being processed. The second costly operation is the second pass that we must perform in the flow texture to make sure that when one section without river particles is found, a suitable value is retrieved. On the other hand, we have the auxiliary texture that contrary to the flow texture, is only affected by the amount of particles used as we load all the particles data into it.

As the number of particles doesn't change abruptly across screen resolutions, we can see in Figure 12 that the difference in values is no bigger than 0.05 milliseconds. As the auxiliary texture only needs to go over all river particles and load their respective values in the texture, this operation can be seen as a linear copy of data

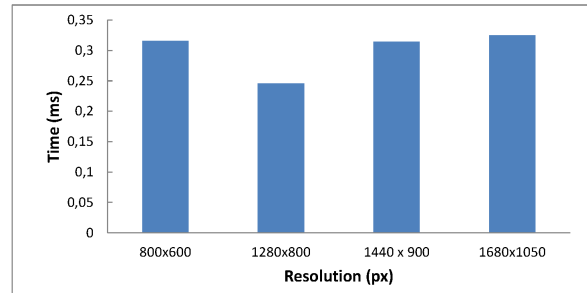


Figure 12: Time taken in milliseconds to load all the data into the auxiliary texture

from the river particles array into the texture, which can be performed quite fast.

4.2 Visualization Engine

As we've previously mentioned the components that make up the Visualization Engine are implemented as two distinct elements: the vertex shader and the fragment shader. Both the Flow Renderer and the Reflection make use of information existent in both of these elements.

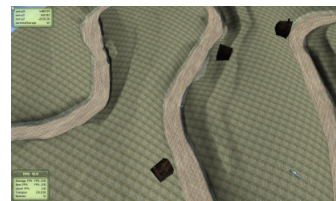


Figure 13: Camera far away from the river surface where little detail can be seen

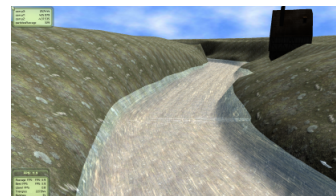


Figure 14: River surface sharing almost the same percentage of screen as all the other elements where several visual details are visible



Figure 15: River occupies almost the entire screen where details can be clearly seen

All the tests were performed with the same river mesh and the camera placed in the positions seen in Figures 13, 14 and 15. This way we can not only understand

how the cost evolves across different resolutions but also how it varies according to different percentages of river mesh present on screen.

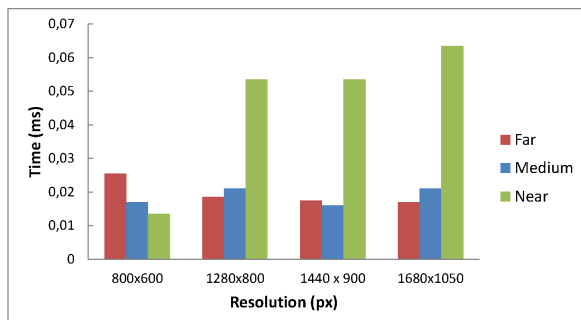


Figure 16: Time taken in milliseconds by the vertex shader to run at different resolutions and different camera distances

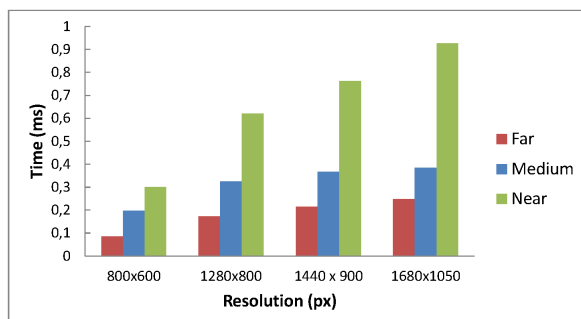


Figure 17: Time taken in milliseconds by the fragment shader to run at different resolutions and different camera distances

4.2.1 Vertex Shader

We have in Figure 16 the results of several measurements performed at different class distances (near, medium and far) and with different resolutions. As most of our computations are performed in the fragment shader, the vertex shader performs only very simple calculations such as transforming vertex positions from one space to another, calculate the camera direction and pass the vertex normal and texture mapping coordinates to the pixel shader. This means that all values are very small and despite the apparent increase in the near distance values when compared with the medium and far values, we see it never reaches differences higher than 0.05 milliseconds.

4.2.2 Fragment Shader

In Figure 17 we can see the time in milliseconds taken by the river fragment shader to complete.

As the resolution increases, the cost of performing the fragment shader increases along with the number of pixels to color. We can also see that the cost increases as we get closer to the river. As the far and medium distances have a smaller amount of river covering the

screen, their costs are much smaller than the near distance which covers almost the entire screen. Despite doing several reads from textures, the cost of running the fragment shader even in the highest resolution is quite small. This is due to the fact that most of the operations we perform are based on reading the information provided by the textures created in the CPU and as far as new calculations go, we perform only the flow algorithm and the reflections which are not very expensive.

4.3 Conclusions and Future Work

We presented a new approach called SiViFlow which simulates realistic rivers in real-time. SiViFlow has two main components: the Simulation Engine and the Visualization Engine. Thanks to the Simulation Engine, SiViFlow is able to adapt to an arbitrary shaped river bed with any number of turns and dynamically calculate the necessary data based on the river surface mesh alone. It also utilizes a concept called river particles to retrieve flow information from the river surface mesh and send it to be drawn in the GPU. The Visualization Engine renders the river flow and is flexible enough to be combined with any visual technique used to simulate water, not being bounded only to the techniques presented in this work. SiViFlow also allows for dynamic objects to alter the course of the flow and change in real-time its behaviour through access to the flow information stored at the river surface mesh. While this approach fulfilled all of the objectives initially defined, there's still room for improvement. With all the advances in the computing capabilities of the new GPU's and respective API's that allow them to perform general computations, a future improvement would be to move the particle update, creation and destruction to the GPU, performing the whole update loop there. As the loading of new data to the flow texture does not have interdependencies among entries, this means that in the limit the whole process of filling the flow texture can be performed completely in parallel. As the approach presented does not have any limitation when it comes to the shading of the water, all visual techniques are compatible with the algorithm and are easily implementable within the Visualization Engine.

4.4 Acknowledgements

This work was supported by national funds through FCT - Fundacao para a Ciencia e a Tecnologia, under project PEst-OE/EEI/LA0021/2013.

5 REFERENCES

- [AMHH08] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman, *Real-time rendering 3rd edition*, ch. Reflections, pp. 386–391, A. K. Peters, Ltd., Natick, MA, USA, 2008.

- [Bri07] Robert Bridson, *Fast poisson disk sampling in arbitrary dimensions*, ACM SIGGRAPH 2007 sketches (New York, USA), SIGGRAPH '07, ACM, 2007.
- [CBL⁺09] Yuanzhang Chang, Kai Bao, Youquan Liu, Jian Zhu, and Enhua Wu, *Particle importance based fluid simulation*, Proceedings of the 2009 Sixth International Conference on Computer Graphics, Imaging and Visualization (Washington, DC, USA), CGIV '09, IEEE Computer Society, 2009, pp. 38–43.
- [CTG10] Jonathan M. Cohen, Sarah Tariq, and Simon Green, *Interactive fluid-particle simulation using translating eulerian grids.*, SI3D, ACM, 2010, pp. 15–22.
- [DG96] Mathieu Desbrun and Marie-Paule Gascuel, *Smoothed particles: a new paradigm for animating highly deformable bodies*, Proceedings of the Eurographics workshop on Computer animation and simulation '96 (New York, USA), Springer-Verlag New York, Inc., 1996, pp. 61–76.
- [DH06] Daniel Dunbar and Greg Humphreys, *A spatial data structure for fast poisson-disk sample generation*, ACM Transactions on Graphics **25** (2006), no. 3, 503–508.
- [EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw, *Animation and rendering of complex water surfaces*, Proceedings of the 29th annual conference on Computer graphics and interactive techniques (New York, USA), SIGGRAPH '02, ACM, 2002, pp. 736–744.
- [Eng03] Wolfgang Engel, *Shaderx shader programming tips and tricks with directx 9*, ch. Rippling Reflective and Refractive Water, pp. 357–362, Wordware Publishing, 2003.
- [Fos96] Nick Foster, *Realistic animation of liquids*, Graphical Models and Image Processing **58** (1996), no. 5, 471–483.
- [GH06] Jostein Gustavsen and Dan Lewi Harkstad, *Visualization of water surface using GPU*, Master's thesis, Norwegian University of Science and Technology, 2006.
- [HKK07] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi, *Smoothed Particle Hydrodynamics on GPUs*, Proceedings of Computer Graphics International, 2007, pp. 63–70.
- [Ney03] Fabrice Neyret, *Advected textures*, Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (Aire-la-Ville, Switzerland, Switzerland), SCA '03, Eurographics Association, 2003, pp. 147–153.
- [Ngu07] Hubert Nguyen, *Gpu gems 3*, ch. Real-Time Simulation and Rendering of 3D Fluids, pp. 633–675, Addison-Wesley Professional, 2007.
- [PF05] Matt Pharr and Randima Fernando, *Gpu gems 2 - programming techniques for high-performance graphics and general-purpose computation*, ch. Octree Textures on the GPU, pp. 595–613, Addison Wesley, 2005.
- [SJ09] Raymon Serway and John Jewett, *Physics for scientists and engineers 8th edition*, ch. The Nature of Light and the Principles of Ray Optics, pp. 1010–1025, Brooks Cole, 2009.
- [Sta99] Jos Stam, *Stable fluids*, Proceedings of the 26th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.
- [Tes99] Jerry Tessendorf, *Simulating ocean water*, SIGGRAPH'99 Course Notes, vol. 2, ACM, 1999.
- [vdLGS09] Wladimir J. van der Laan, Simon Green, and Miguel Sainz, *Screen space fluid rendering with curvature flow*, Proceedings of the 2009 symposium on Interactive 3D graphics and games (New York, NY, USA), I3D '09, ACM, 2009, pp. 91–98.
- [vH11] Frans van Hoesel, *Tiled directional flow*, ACM SIGGRAPH 2011 Posters (New York, USA), SIGGRAPH '11, ACM, 2011, pp. 19:1–19:1.
- [WLL04] Enhua Wu, Youquan Liu, and Xuehui Liu, *An improved study of real-time fluid simulation on gpu: Research articles*, Computer Animation and Virtual Worlds **15** (2004), no. 3-4, 139–146.
- [YN08] Qizhi Yu and Fabrice Neyret, *Models of animated rivers for the interactive exploration of landscapes*, Ph.D. thesis, Institut National Polytechnique de Grenoble, November 2008.
- [YNBH09] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch, *Scalable real-time animation of rivers*, Computer Graphics Forum (Proceedings of Eurographics), vol. 28 (2), March 2009.

Simplification of 3D Point Clouds sampled from Elevation Surfaces

Van-Sinh NGUYEN
Aix-Marseille University
CNRS, Laboratory LSIS
UMR 7296, Marseille, France
Tel: 33.4.91.82.85.28
van-sinh.nguyen@univ-amu.fr

Alexandra BAC
Aix-Marseille University
CNRS, Laboratory LSIS
UMR 7296, Marseille, France
Tel: 33.4.91.82.85.32
alexandra.bac@univ-amu.fr

Marc DANIEL
Aix-Marseille University
CNRS, Laboratory LSIS
UMR 7296, Marseille, France
Tel: 33.4.91.82.85.25
marc.daniel@univ-amu.fr

ABSTRACT

This paper introduces a new technique to simplify a 3D point cloud sampled from an elevation surface and organized in voxels. The method consists of three steps: in a first step, the boundary of the surface is extracted and simplified; in a second optional step, we roughly simplify the surface inside its boundary; in a third step, we present an elaborate method for simplification while keeping its boundary. Our method preserves the distribution of points, the initial geometry and characteristics of the surface, even with high simplification rates.

Keywords

Boundary Extraction, Boundary Simplification, Surface Simplification, Principal Component Analysis (PCA).

1 INTRODUCTION

Simplification of a 3D point cloud belonging to a surface is an important steps in geometric modeling and surface processing. The purpose of surface simplification of a 3D point cloud is to reduce the number of points, save the memory, improve the effect of computation and optimize the processing of the geometric model. During simplification, the original shape of the surface must be kept, without shrinking or deformations.

Nowadays, the modern 3D acquisition and modeling technology allow producing a large amount of point samples from real-world objects. Different existing researches (and especially for meshes) are available for processing of the continuous surfaces, but the case of 3D point clouds simplification remains a challenging issue.

Our problem originates in the questions of processing large 3D point clouds issued from a seismic data (themselves extracted from a 3D sparse volume [Philippe09]). The seismic acquisition does not permit to measure all the points in the 3D volume, explaining the fact that the 3D volume is sparse. The 3D points are actually stored in a voxel structure in this volume

(each voxel is considered as a 3D point, and has three real coordinates xyz), hence implicitly the 3D volume contains neighboring information even in a sparse context.

Most existing approaches have a common drawback: in the case of open surfaces (that is surfaces with boundaries), simplification induces a shrinking of the surface. Hence, in order to preserve the initial shape, our approach starts by an extraction and simplification step of the boundary. In a previous work, we have proposed a method for extracting and simplifying the boundary of a surface [Sinh12]. The present paper continues this work and introduces a method to simplify the inside of this surface. To handle potentially huge clouds, our method consists of two steps: an optional initial rough simplification (basically designed to adjust the sampling rate) followed by a more elaborated simplification step. As the point cloud is sampled from elevation surfaces, points are first projected onto a 2D grid in xy plane to process with the first step, while the second step is directly processed in the 3D grid.

The remainder of this paper is organized as follows: in section 2, we present work related to surface simplification of a 3D point cloud. We present our method in detail, which includes problem analysis, building the criteria and implementing the algorithms in sections 3. The results and evaluation of our method are presented in sections 4 and 5. The last section is our conclusion.

2 RELATED WORKS

Different existing methods which have been studied and developed are not only applied to sim-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ply the surface of 3D point clouds, but also applied to simplify the surface of triangular mesh [Garland99, Pauly02, Van06, HaoSong09, Zhe07]. Among them, PCA (Principle Components Analysis) is a popular tool, a well known method that can be used to simplify the surface of 3D point clouds [Mederos04, Yu06, Alexandra07, David08].

Garland et al (1999) [Garland99] developed an algorithm to simplify the surface of a polygonal model based on the iterative contraction of vertex pairs. Starting from the initial model M_1 , an edge e_{v_1, v_2} will be contracted to a new position \bar{v} if the distance $\|v_1 - v_2\| < \text{threshold}$. The process is repeated until the simplification goals are satisfying. The last model M_2 approximates M_1 . In order to preserve the shape of the surface and optimize the placement of vertices after contraction, the authors used the quadric error matrices to track the approximate error of the model. This method is time and memory demanding, but it avoids distortion of the original shape. However, evaluation of the quadratic error metric is closely related to the mesh structure (and to the face neighborhoods). Hence, it cannot easily be adapted in our setting.

Pauly et al (2002) [Pauly02] introduced, analyzed, compared and implemented a number of methods to simplify the surface of 3D point clouds. One of these methods is called "Clustering". The surface of 3D point clouds is clustered by splitting it into a subset of points; then, replace all points in each cluster by one representative point. This region-growing is terminated when the size of the cluster reaches the maximum bound. This method leads to simplifying the surface effectively. However, each cluster is a sphere with a radius α on the surface. Therefore, the points outside these clusters are not simplified completely after the iterative processing.

Boris et al (2004) [Mederos04] proposed a method to reconstruct and smooth a surface from noisy point clouds. At first, he smoothed the original point clouds to reduce the noisy points by using a robust projection procedure, while keeping the shape of the surface. The next step, data of 3D point clouds are clustered by partitioning into a subset of clusters. Then, he applied PCA to analyze, reduce the size of the original points, and determine a representative point for each cluster. In the next step, a triangular surface is obtained from the representative points of each cluster to obtain a rough surface which approximates the original surface. The last step, this rough surface is refined to get an optimal one. This is a complete method for surface reconstruction of a point cloud. However, the computing is complex during projecting, clustering, reducing, meshing and refining the point clouds, leading to a computation heavy and costly.

Normally, to simplify the surface of 3D point clouds, the existing approaches aim to cluster a subset of

points, and then grow on the surface to simplify. The problem is how to determine the neighboring points in a local region of the surface. Y.J Zhang et al (2010) [Zhang10] proposed a way to define the nearest neighbouring points by using a cylinder. The points are dropped into a bounding cylinder based on the specified threshold (the radius of the cylinder); then, they are projected on the line as its center axis to simplify the points inside. The same as method [Pauly02], for each iterative step, the outside points are not simplified completely.

Frey et al (2007) [Frey07] presented a method (the "affinity propagation") to cluster by passing messages between data points. This method measures the similarity of each point-pair of the input data points. Each point in a point set is assigned as a node of a network, the real-valued messages are exchanged between data points (nodes) along the edge of the network until a high-quality set of exemplars corresponds to the cluster which gradually emerge. However, the cost of computing is expensive because the transmission process between the points is computed recursively.

Jae-Young et al (2005) [Jae05] and Tamal et al (2011) [Tamal11] introduced a method by using an octree partitioning to divide the point clouds into a small subset, then process on each subset as a node of an octree on 3D space and quadtree on the 2D grid. At first, a root node of a point cloud is divided into four in 2D or eight in 3D. Then, the child nodes are recursively divided until satisfying the condition of the threshold. After that, each node can be considered as a point during the simplification.

Morales et al (2010) [Morales10] suggested a method to smooth and decimate the points from an unstructured point cloud based on the radial based function (RBF). The points are computed based on the kd-tree nearest neighbors. Starting from a seed point p_i , the neighboring points (p_n) are calculated by an Euclidean distance $\|p_i - p_n\|$ to determine the radius r . All points within r are mapped from a 3D point set to the 2D space; the point set components are mapped into each axis plane on each square matrix $M \times M \times 3$ in domain N_{ix}, N_{iy}, N_{iz} . The next step is using a convolution Gaussian Kernels function $C = M \otimes G(\mu, \sigma^{d(k)})$ for each axis N_{ij} to smooth and estimate the new center point in each component $p'_{x,y,z}$. Finally, the 3D point sets are smoothed and simplified according to the local surface features.

As we have described and analyzed, the above methods are suitable for dispersive data or unorganized point clouds but lead to an expensive computation. In our work, we take advantage on the structure of voxels and their neighborhood information. We can adapt these methods to simplify the surface efficiently; preserve the shape and point distribution of the surface.

3 OUR METHOD

Our method consists of three steps (see figure 1). The first step (boundary extraction and simplification), we have presented in the previous work [Sinh12]. In this paper, we present the second and the third step for simplifying the surface inside its boundary.

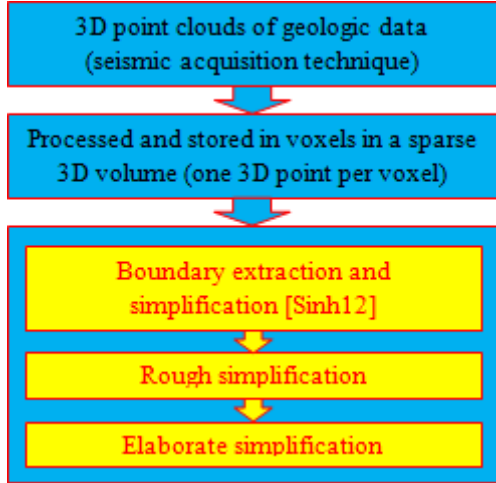


Figure 1: A method for simplifying the surface.

It is interesting to summarize the main idea of the method we applied first to extract and simplify the border of a surface [Sinh12]. We define a method to extract the boundary based on k -square neighborhood of each point up to a fixed integer distance k . Our algorithm starts from an initial boundary point of the surface; then, an exterior boundary is built point by point by iteratively computing the next point via growth functions. After that, we build an algorithm to simplify this boundary by first study the alignment of points and second study the variation of elevation. In our method, the complexity of algorithms is proved more efficient than existing methods. Moreover the initial shapes of the surface are also preserved for the simplification step since the boundaries are kept.

3.1 Rough simplification

3.1.1 Overview

Rough simplification is a preliminary step designed to handle large point clouds: points are imported in a fine regular grid and each non empty voxel is replaced by a single representative vertex. Hence, the goal of this step is merely to adjust sampling density. In this algorithm, 3D point clouds (organized in a sparse 3D regular grid) are first projected onto the 2D grid in the x, y plane. This 2D point cloud (set of non empty voxels) is subdivided according to a regular grid of size s (this size is defined by the user according to the desired final sampling rate) (see figure 2a). Then, each non-empty cell is replaced by a single representative point: the barycenter of contained points. This step, even if rough, can be

justified in terms of resolution: it is merely a resolution adaptation (in case the resolution of the data is too high compared to the expected results). The important point in this step is that we will not simplify boundary points (as they have already been handled in the previous work [Sinh12]); and this step should be applied using a small size of cells in order to avoid distorting the surface.

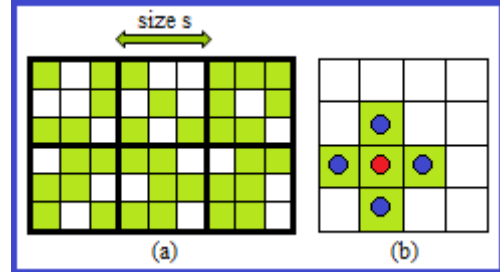


Figure 2: a) The size of a cell. b) The barycenter of the points (red color) in the cell.

3.1.2 Notation

In the sequel, we use the following notations:

- G : the 2D initial regular grid,
- C : the regular grid of size s built over G ,
- S : the subset of cells in C which are non empty,
- S_q : a cell on the 2D grid belonging to S ,
- p_q : barycenter of the points included in S_q .

3.1.3 Algorithm

As the size of the cells is small and as we want to preserve boundary points, if a cell contains boundary points, no further representative vertex will be inserted, only included boundary points are kept. Otherwise, if a cell does not contain boundary points, we compute the barycenter of the points in this cell. Based on the above description, we propose a very simple algorithm (Algorithm 1) with a linear complexity to roughly simplify the surface.

Algorithm 1 roughSimplification(s)

```

1: for each cell  $S_q \in S$  do
2:   if  $S_q$  contains boundary points then
3:     keep only boundary points;
4:   else
5:     replace all points by  $p_q$ ;
6:   end if
7: end for

```

3.2 Elaborate simplification

3.2.1 Overview

In this step, we focus on two main points to process the surface: curvature of the surface and point density. We

process the surface directly in the 3D grid. As previously, the sparse 3D grid (equivalent to the point cloud) is divided according to a regular 3D grid C . The initial size of the cells of C is large (defined by the user) and elaborate simplification will further subdivide cells of C according to density and curvature criteria. If cells contains boundary points, they are processed based on the combination between boundary density and local curvature in these cells. Otherwise, subdivision is based on local curvature within each cell and adapted to the size of neighboring cells. After simplification, the distribution of points has to vary continuously; it must be constrained regularly from the exterior boundary to the inside of the surface. This constraint is introduced to avoid creating bad triangles (in the sense of Delaunay triangulation) in a further meshing step.

3.2.2 Analysis

Obviously, our rough preliminary simplification is too basic to reach high simplification rates. It is useful only to adjust the resolution or as a first decimation for huge point clouds (for which a more elaborate simplification cannot be applied directly because of time and space complexity issues). Hence, this preliminary step is optional.

In the case of complex surfaces with a high curvature, simplification must be based both on density and curvature criteria. For this reason, we develop an advanced algorithm to simplify the surface more elaborately. This algorithm is based on an octree subdivision of the surface adapted to its curvature, point density and to the border density. We will combine two subdivision criteria to simplify the surface: subdivision according to the boundary density and subdivision according to the curvature.

3.2.3 Subdivision according to the boundary density

An important issue is that point density should vary "smoothly" (in order to preserve the shape of triangles in a further meshing step). It must be constrained continuously on the surface and propagate regularly from the boundary to the inside of the surface. Therefore, in this paper we propose a method to simplify the surface inside its boundary. In order to subdivide cells according to the boundary density, we have to build a subdivision criterion. At first, we analyze the density of boundary points (number of boundary points in a cell) and their distribution. Our criterion is based on the size of a cell, the number of boundary points and the distance between them.

a) Notation and formula construction

We will use the following notations:

- C_q : a cell (size s) in the 3D grid,
- Nbp : the number of boundary points in C_q ,
- d_{max} : the maximum distance between two boundary points in C_q ,
- L_s : the level of subdivision of a cell (see figure 3),
- s' : the size of the smaller cells after each subdivision of C_q : $s' = \frac{s}{2^{L_s}}$.
- p_i, p_j : point i^{th} , point j^{th} of C_q .

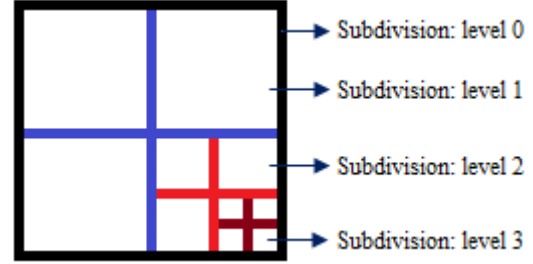


Figure 3: The level of subdivision in a cell.

In our context, data points are organized based on a 3D grid structure, each point in a cell has xyz coordinates and in the sequel, we will use the Euclidean distance to compute the distance between points. Hence the maximum distance between boundary points in a cell is given by:

$$d_{max} = \max_{i, j \in (1..Nbp); i \neq j} (\|p_i - p_j\|) \quad (1)$$

b) Boundary density criteria

Subdivision according to boundary density is performed from cells containing boundary points (called first ring) towards the surface interior (ring by ring, starting from the boundary). In the sequel, we will denote by r_i the i^{th} ring of cells based on the 8-connectivity (hence, r_1 is the set of boundary cells). There is a relationship between the density of points and the distance between them in a cell. Obviously, as the density of boundary points in a cell increase, the distance between them will decrease. The formula: $D(density) = Np(number\ of\ points)/V(volume)$ can be applied to compute the density of points on a volume. In our case, we focused on the number of boundary points Nbp in a cell and its size s to calculate point density PD of that cell ($PD = Nbp/s$). Hence our criterion is based on PD and d_{max} :

$$(PD > threshold_{pd}) \text{ and } (d_{max} > threshold_d) \quad (2)$$

In order to preserve the shape of the surface for a further triangular meshing step, the size of cells must vary smoothly. Therefore, for boundary cells (also called

first ring cells), we state a specific subdivision criterion: if a cell C_q (containing boundary points) satisfies the first condition (2), then we check the size of C_q . If the size is less than or equal than a threshold, we keep only boundary points; else, we keep boundary points and the barycenter of inner points in that cell. Otherwise, C_q is subdivided (as an octree).

Starting from the second ring (which contains inner points of the surface), we subdivide cells both according to the local curvature and previous ring cell sizes. Cells are processed ring by ring from the outside to the inside of the surface. The cell size in ring r_i is subdivided according to the sizes of neighboring cells of ring r_{i-1} (the outside adjacent ring of r_i). It means that, if an inner cell satisfies the curvature criterion, we subdivide it according to the average subdivision level of all nearest neighboring cells. Let $C_q \in r_i$ and let $\{C_1^{i-1}, \dots, C_m^{i-1}\}$ be the set of neighboring cells in r_{i-1} , the subdivision level of C_q is computed as:

$$size(C_q) = \frac{1}{m} \sum_{j=1}^m size(C_j^{i-1}) \quad (3)$$

In the end, the cell size varies smoothly; and if the curvature inside a cell is low, all points in this cell are replaced by one representative point. In next section, we build a flatness criteria in order to subdivide cells according to their curvature.

3.2.4 Subdivision according to the curvature

Our goal is to preserve the shape of the surface after simplification. In this part we process the cells containing inner points, from the second ring to the inside of surface. For each cell we apply a principal component analysis (PCA) to estimate the average local curvature of the surface. We thus define a flatness criterion and subdivide cells accordingly.

a) PCA flatness criteria

PCA can be used as a useful statistical method to analyze data. This is a technique that can be applied to simplify a surface of 3D point clouds (see [Pauly02, Mederos04, Alexandra07, Zhe07, MZhang11]). In order to estimate the curvature/flatness of a cell, we compute the PCA of the vertices of the cell. The eigenvalues of the corresponding covariance matrix provide a curvature information and we define accordingly a flatness criterion. Cells that do not meet this flatness criterion are subdivided until either their size is lesser than a threshold or they satisfy the criterion.

We use the formula below to compute the covariance matrix for each cell:

$$C = \frac{1}{N} \sum_{i=1}^N (p_i - \bar{p})(p_i - \bar{p})^t; \quad (4)$$

Where:

- N : a set of points in each C_q ,
- \bar{p} : barycenter of points in C_q ,
- λ_i, v_i : the i^{th} eigenvalue and i^{th} eigenvector of C .

The eigenvectors of C provide information about the principal directions of a point set. More precisely, the eigenvectors provide main axes of the cloud, while eigenvalues provide its stretching along the corresponding axes. Hence, the eigenvector associated to the smallest eigenvalue provides an average normal vector while both other eigenvalues are related to principal curvatures.

Following the above analysis and applying the ideas introduced in [Pauly02, Mederos04, David08, MZhang11], let us sort eigenvalues: $\lambda_0 \leq \lambda_1 \leq \lambda_2$. If the value of λ_0 is very small or even equal 0, that means all the points in a cell are approximately on a plane (it satisfied the flatness criteria). In such a case, the average normal vector on a local surface within a cell can be determined based on the direction of v_0 . The **flatness criterion** " ∂ " below is considered as a condition to further subdivide cells (and hence to control the simplification of the surface):

$$\partial = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \quad (5)$$

For each point on the local surface, if their normal vectors are distributed isotropically, these points will lie on the same plane. This solution is given by Hugues Hoppe [Hoppe92] to compute the orientation of the tangent plane: for each data point p_i , a tangent plane is computed by least-squares approximation based on PCA of the k nearest neighbor of p_i .

In our case, we use the flatness criteria(5) to estimate the local curvature in a cell. The minimum value of ∂ equal 0, while its maximum value equal 1/3, and our flatness criteria is based on the range of these values. (see figure 4)

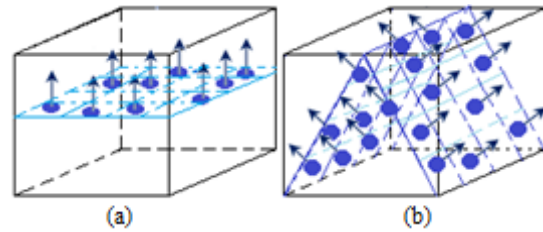


Figure 4: Estimation of the curvature in a cell: (a) The points are approximately on a plane within a cell (λ_0 is very small, λ_1 and λ_2 are large); (b) λ_0 is large or ($\lambda_0 \simeq \lambda_1 \simeq \lambda_2 \simeq 1$) or ($\partial \simeq 1/3$) \Rightarrow this cell is subdivided.

The curvature in a cell is first determined by computing ∂ . Then, ∂ is compared with a threshold value from the user. If $\partial \leq threshold_{\partial}$, we replace all points in this cell by one representative point. This way can simplify

the surface efficiently and the ratio of simplification is very high (if the points in that cell are approximately on a plane). However, the density of points could vary irregularly after a large number of points have been removed. For this reason, we have to combine with the computation of point density and size of cells to constrain the distribution of points on the surface to be as regular as possible.

3.2.5 Algorithm

According to the previous analysis, we now define our simplification algorithm. Our algorithm covers cells ring by ring (starting from boundary cells), each ring is processed clockwise (see figure 5).

We start from the first ring, blue color (i.e. the ring of boundary points). In this ring, we begin with the left-most cell (1) and follow the clockwise direction to compute, subdivide and simplify each cell. From the second ring (yellow color), we also begin with the left-most cell (2) and so forth for following rings (third - green, fourth - pink, etc). The algorithms below are used to simplify the surface: algorithm 2 is used to process the cells containing boundary points in the first ring.

Algorithm 2 SimplifyBoundaryCells(s)

```

1:  $Nbp = 0, L_s = 0$ ; //start from the left-most cell, follow the clockwise direction.
2: for each boundary cell  $C_q$ (size  $s$ )  $\in S$  do
3:   compute  $Nbp, d_{max}$ ;
4:   if  $C_q$  satisfy the density criteria(2) then
5:     if size  $s \leq threshold_s$  then
6:       keep only boundary points;
7:     else
8:       replace all points by boundary points and the barycenter of inner points;
9:     end if
10:    else //subdivide  $C_q$  by  $L_s$ .
11:       $L_s = L_s + 1$ ;
12:       $s' = s / (pow(2, L_s))$ ;
13:      for each  $C_q(s') \in C_q(s)$  do
14:        if  $C_q(s')$  contains boundary points then
15:          SimplifyBoundaryCells( $s'$ );
16:        else
17:          SimplifyInnerCells( $s'$ );
18:        end if
19:      end for
20:    end if
21: end for

```

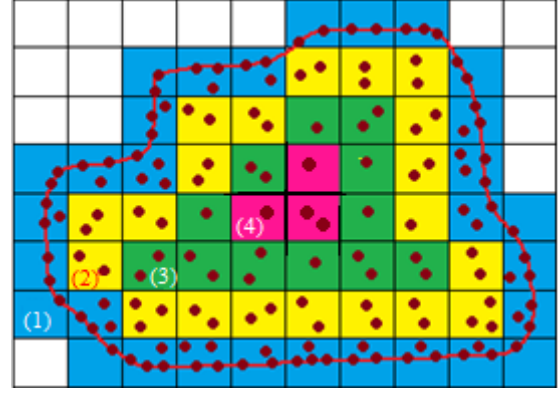


Figure 5: Illustration of the elaborate algorithm.

Algorithm 3 is used to process the cells containing inner points from the second ring to the inside of surface.

Algorithm 3 SimplifyInnerCells(s)

```

1:  $L_s = 0$ ; //start from the left-most cell, follow the clockwise direction.
2: for each inner cell  $C_q$  (size  $s$ )  $\in S$  do
3:   compute the covariance matrix of points in  $C_q$ ;
4:   if  $C_q$  satisfy the flatness criteria(5) then
5:     subdivide  $C_q$  based on (3);
6:     replace all points by the barycenter in each sub-cell;
7:   else //subdivide  $C_q$  by  $L_s$ .
8:      $L_s = L_s + 1$ ;
9:      $s' = s / (pow(2, L_s))$ ;
10:    SimplifyInnerCells( $s'$ );
11:   end if
12: end for

```

For each inner cell, we compute the curvature criterion (5). If it satisfies the threshold, we first subdivide this cell based on (3); then, replace all points in each sub-cell by their barycenter. Otherwise, we subdivide this cell and repeat the process until all conditions of the criterion are satisfied.

In this step, our computing experiences have seen that the processing time mostly depends on values of ∂ ; before and after combining with step one (rough simplification) (see table 2), and less depends on s (size of a cell). Normally, the number of points in a cluster (using PCA) is around from 30 points [Carsten04, RenFang08, Morales10]. In our case, the curvature within a cell of a geologic surface is low and the 3D points are sparse. Therefore, we choose $s \leq 10$ (that is initial cells containing at most 100 voxels) and many values of ∂ to implement. As a result, the time is affected if the number of points in a cell greater than 36 or ∂ close to 0 and before combining with step one. We keep the boundary and combine two steps (rough and elaborate) to simplify a surface; thus, the surface is simplified com-

pletely, the initial shapes of the surface are preserved, and the time is controlled.

4 RESULTS

In this section, we present some of our results. For the step one (rough simplification), the computation are very fast. The algorithm has been tested on many surfaces with different number of points to compare the running time and simplification rate with an existing method (cluster vertices) [Pauly02]. The results are presented in table 1, the running time of our method is faster than the clustering method, while the simplification rate is slightly lower (depending on the initial shapes of input surface) because we kept the boundary points.

Input points	Our method		Cluster method	
	Output points/s.rate	time (ms)	Output points/s.rate	time (ms)
32402	1881/94%	36	1075/96%	303
68956	3695/95%	53	2432/96%	544
148317	6368/96%	98	4675/97%	1149
346796	13030/96%	206	11068/97%	2766
664582	22388/96.6%	377	19872/97%	5739
1006712	67360/93%	651	28850/97%	8501

Table 1: The comparison between our method (rough simplification) and clustering method. We use the same size of a neighboring distance between the points, and run on the same computer (s.rate: simplification rate; ms: millisecond).

In this step, the simplification rate is controlled by the cell size. In our method, although boundary points are kept to preserve a part of the shape of surface, this approach does not take into account the curvature of the surface and hence is too rough to be applied with high simplification rates. If we use a larger size of cells to simplify, the received results are not accurate (see figure 7). Therefore, this step can only be applied to simplify a simple surface of 3D points or to adjust the resolution of a 3D point cloud by using a small size of cell. In the clustering method, all points of the surface (boundary points and inner points) are simplified; the shape of the output surface is not well preserved (see figure 8).

In step two (elaborate simplification), we have tested our approach on different surfaces with different numbers of 3D points and different values for ∂ . The results are detailed in table 2. We provide the values of ∂ in order to show that: if the value of ∂ is close to 0, the obtained surface is smooth, close to the initial surface (small simplification rate) and the processing time is low; otherwise, if the value of ∂ is close to 1/3, the obtained surface is far from the original one (higher simplification rate) and the running time is higher. However, we have maintained boundary points, and constrained the point distribution from the boundary to the

inside of the surface. Therefore, we have obtained the output surfaces preserving the initial geometry of the surface (see figure 10). Figure 9 shows the result of the point distribution constrained from the boundary to the inside of the surface. As a result, a good triangular surface can be obtained in a further meshing step.

P.input (kb)	Values of ∂	Time1 (ms)	Time2 (ms)	P.output (s.rate)
60511 (976)	$\partial \leq 0.03$	5271	3231	9879/84%
	$\partial \leq 0.12$	5026	2958	9377/84.5%
	$\partial \leq 0.20$	3776	2910	6786/89%
148317 (2461)	$\partial \leq 0.03$	22106	14194	21122/86%
	$\partial \leq 0.12$	21167	13825	20820/86%
	$\partial \leq 0.20$	15896	12079	18916/87%
346796 (5727)	$\partial \leq 0.03$	114795	111362	56448/84%
	$\partial \leq 0.12$	111289	107309	52187/85%
	$\partial \leq 0.20$	110623	101544	50112/86%
866639 (14500)	$\partial \leq 0.03$	832865	191153	147328/83%
	$\partial \leq 0.12$	786980	185491	138622/84%
	$\partial \leq 0.20$	581159	166116	112633/86%

Table 2: The running time of step two before (Time1) and after (Time2) combining with step one; the simplification rate (s.rate) after using the same size of cells; different values of ∂ (kb: kilobyte; ms: millisecond).

5 EVALUATION

Our method has two advantages compared to existing methods. First, we use a cell to gather and compute the points in a local neighborhood to simplify the surface. By using a cell, there are no outside points between the cells; only one loop is used to consider all points of the surface. On the contrary, the other methods [Pauly02, Zhang10, Morales10] use a sphere or a cylinder (both are the same) to compute the neighboring points within a threshold value of a radius r (see figure 6). Therefore, after each iterative operation, they have to process the points outside of these sphere/cylinder. The second advantage is that searching to compute a neighboring point within a cell is faster than within a sphere [Matthew96]. Our approach also takes advantage of the fact that our data are already organized in a sparse numeric volume, and hence we don't need to lose time and memory space to build accelerating data structure for k_neighbors computation (such as kd-trees or octrees).

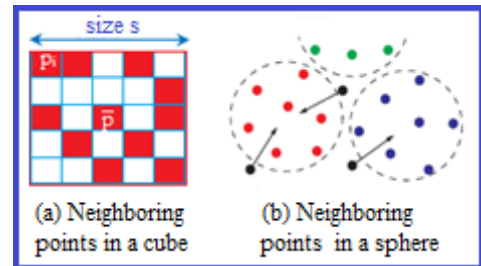


Figure 6: Determining of a neighboring point.

6 CONCLUSION

In this paper, we have presented a method to simplify an elevation surface defined by a 3D point cloud. It is a part of our research in the field of geometric modeling of oil reservoir. The input data are a mass of 3D point clouds, and the number of points can reach millions of points. Therefore, our first approach focuses on data processing and surface simplification. Successively, we succeed in boundary extraction and simplification of the surface, while preserving the original shape of the surface as expected [Sinh12]. The surface simplification of 3D point clouds using PCA can normally yield an expensive computation. In our case, the input data are stored in the 3D grid volume, implicitly containing the neighborhood information for each point. We have taken this advantage; combined two steps for rough and elaborate simplification; and two ways of subdivision by using a cell to grow and simplify the surface. The output surface preserves the initial shape of the input surface, the point density and the point distribution are kept regularly, constrained from the boundary to the inside of surface. This good distribution of points is an advantage to obtain a good triangulation of the point clouds. Obtaining this triangulation by a fast method corresponds to our forthcoming work.

7 ACKNOWLEDGMENTS

The work reported in this article is a part of a PhD thesis, in which the finance is supported by the cooperation between Vietnam's government (MOET) and France's government (Campus France). We would like to thank all their valuable helps. We would also thank the reviewers for their valuable comments.

8 REFERENCES

- [Sinh12] Van-Sinh NGUYEN, Alexandra BAC, Marc DANIEL, "Boundary Extraction and Simplification of a Surface Defined by a Sparse 3D Volume", *Proceeding of the third international symposium on information and communication technology SoICT 2012*, Pages. 115-124, ACM-ISBN: 978-1-4503-1232-5, August 23-24, Vietnam, 2012.
- [Van06] Nam-Van TRAN, "Traitement de surfaces triangulées pour la construction des modèles géologique structuraux", *PhD Thesis*, Université de la Méditerranée, 2008.
- [Garland99] Michael Garland, "Quadric-Based Polygonal Surface Simplification", *PhD Thesis*, Carnegie Mellon University, 1999.
- [Philippe09] Philippe Verney, "Interprétation géologique de données sismiques par une méthode supervisée basée sur la vision cognitive", *PhD Thesis*, École Nationale Supérieure des Mines de Paris, 2009.
- [Carsten04] Carsten Moenning, Neil A. Dodgson, "Intrinsic point cloud simplification", *International Conference Graphicon '14*, Moscow, Russia, 2004.
- [RenFang08] Ren-fang WANG, Wen-zhi CHEN, San-yuan ZHANG, Yin ZHANG, Xiu-zi YE, "Similarity-based denoising of point-sampled surfaces", *Journal of Zhejiang University SCIENCE A*, Volume. 9, Number. 6, Pages. 807-815, 2008.
- [Alexandra07] A.Bac, V.Tran Nam, M.Daniel, "A hybrid simplification algorithm for triangular mesh", *Graphic Conference 2007*, Pages. 17-24, Moscow.
- [Alexa01] M.Alexa, J.Behr, D.Cohen-Or, S.Fleishman, D.Levin, C.T.Silva, "Point Set Surfaces", *Proceedings of the conference on Visualization '01*, San Diego, CA, USA - October 2001.
- [Cignoni98] P.Cignoni, C.Rocchini, R.Scopigno, "Metro: Measuring error on simplified surfaces", *The Eurographics Association 1998*, Volume. 17, Number. 2, June 1998.
- [Matthew96] Matthew T. Dickerson, David Eppstein, "Algorithms for proximity problems in higher dimensions", *Journal Computational Geometry, Theory and Applications* Pages, Volume. 5, Pages. 277-291, 1996.
- [Pauly02] M.Pauly, M.Gross, L.P.Kobbelt, "Efficient Simplification of Point-Sampled Surfaces", *Visualization, 2002. VIS IEEE*, ISBN: 0-7803-7498-3, Pages. 163 - 170, Boston, MA, USA, 2002.
- [Moenning03] Carsten Moenning and Neil A. Dodgson, "A New Point Cloud Simplification Algorithm", *In Proceedings 3rd IASTED Conference on Visualization, Imaging and Image Processing*, Pages. 1027-1033, Spain, 8-10 Sep 2003.
- [Mederos04] B.Mederos, L.Velho, L.H.Figueiredo, "Smooth Surface Reconstruction from Noisy Clouds" *Journal of the Brazilian Computer Society*, Volume. 9, Number. 3, Pages. 52-66, ISSN: 0104-6500, Campinas Brasil, Apr. 2004.
- [Zhang10] Y.J.Zhang, L.L.Ge, "A Robust and Efficient Method for Direct Projection on Point-sampled Surface", *International Journal of Precision Engineering and Manufacturing*, Volume. 11, Number. 1, Pages. 145-155, DOI: 10.1007/s12541-010-0018-z, 2010.
- [Morales10] R.Morales, Y.Wang, Z.Zhang, "Unstructured Point Cloud Surface Denoising and Decimation Using Distance RBF K-rearest Neighbor Kernel", *Proceedings of the Advances in multimedia information processing*, ISBN: 3-642-15695-9 978-3-642-15695-3, China, 2010.
- [HaoSong09] Hao Song, Hsi-Yung Feng, "A progressive point cloud simplification algorithm with

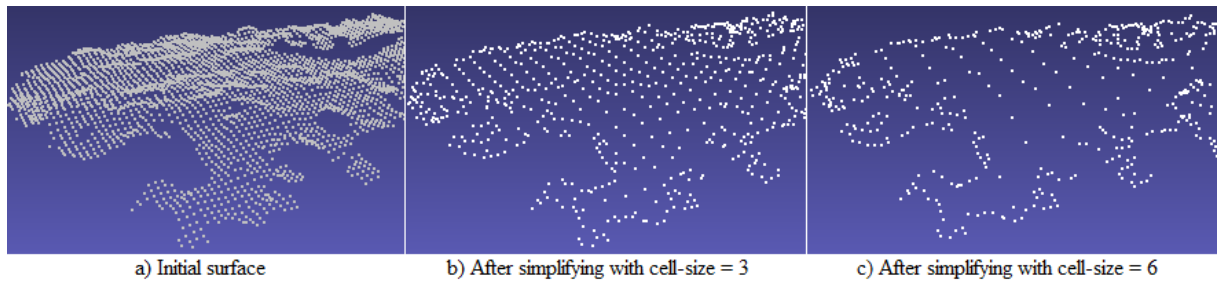


Figure 7: Rough simplification: the shape of the initial surface is not preserved and received results are not accurate using a large cell-size (c).

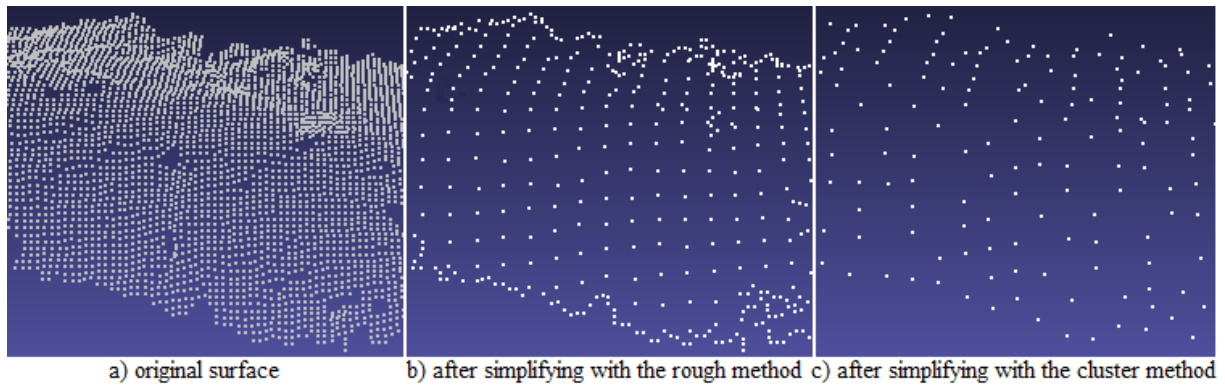


Figure 8: Shape comparison (computing the approximate error) by using the rough simplification method (Max: 0.014235, Mean: 0.000486) and the cluster method (Max: 0.029596, Mean: 0.000994), with the same size of neighboring distance.

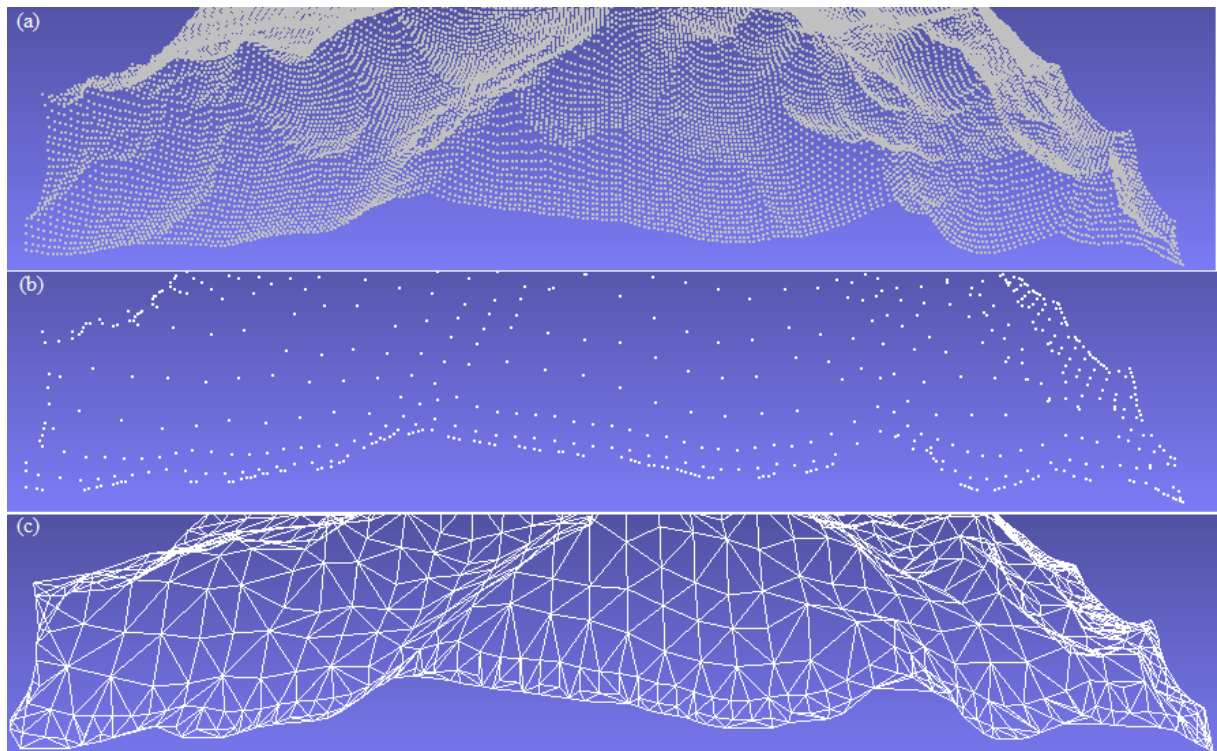


Figure 9: a) An elevation surface of 3D data points (66049 points); b) After simplifying by using the elaborate method (cell-size=8, $\partial \leq 0.09$, remaining points: 1840), the point distribution are constrained from the boundary to the inside of the surface; c) A good triangular surface can be obtained in a further meshing step (the approximate error between (a) and (c) is Max: 0.061598; Mean: 0.035884)

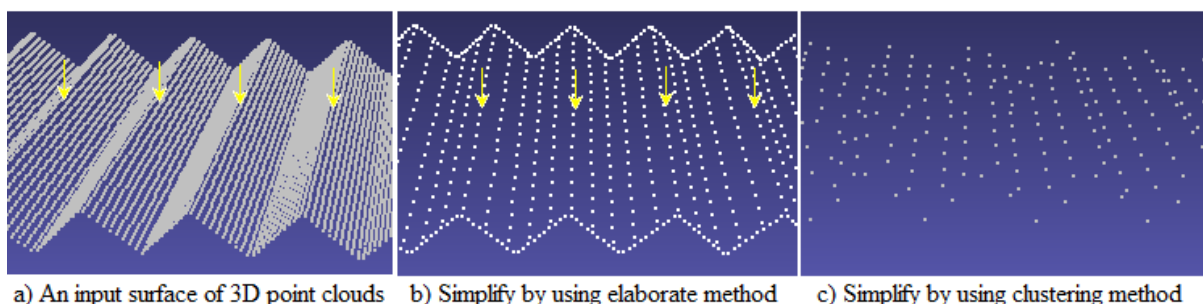


Figure 10: Comparison the shape of the surface between two methods by using the same size of neighboring distance. (a) An input surface of 23559 points; (b) After using elaborate simplification method (time: 858 msec, remaining points: 2305), the curvature (sharp lines: yellow arrows) of the surface is maintained; (c) After using clustering method (time: 255 msec, remaining points: 801), the curvature of the surface is not preserved.

- preserved sharp edge data", *Technology - INT J ADV MANUF TECHNOL*, Volume. 45, Number. 5, Pages. 583-592,, 2009.
- [Hoppe92] Hugues Hoppe, Tony DeRose, Tom Duchampy, John McDonaldz, Werner Stuetzlez, "Surface reconstruction from unorganized points", *Proceeding SIGGRAPH '92 Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, Pages, 71-78, Volume 26 Issue 2, USA, 1992.
- [Yoo09] D.J.Yoo, H.H.Kwon, "Shape Reconstruction, Shape Manipulation, and Direct Generation of Input Data from Point Clouds for Rapid Prototyping", *International journal of precision engineering and manufacturing*, ISSN: 2005-4602, Volume. 10, Number. 1, Pages. 103-113, 2009.
- [Frey07] B.J. Frey, D.Dueck, "Clustering by Passing Messages Between Data Points", Volume. 315, Number. 5814, Pages. 972-976, 2007.
- [Mario09] Mario Richtsfeld, Markus Vincze, "Point Cloud Segmentation Based on Radial Reflection", *CAIP '09 Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, ISBN: 978-3-642-03766-5, Volume. 5702, Pages. 955-962, Berlin, 2009.
- [Mederos03] B.Mederos, L.Velho, L.H.Figueiredo, "Robust Smoothing of Noisy Point Clouds", *Conference on Geometric Design and Computing*, *ACM Trans on Graphics* 22, Pages. 4-32, 2003.
- [MZhang11] M.Zhang, N.Anwer, L.Mathieu, H.B.Zhao, "A Discrete Geometry Framework for Geometrical Product Specifications", *CIRP Design Conference*, Pages. 142-148, South Korea, March 2011.
- [Franc01] M.Franc, V.Skala, "Triangular Mesh Decimation in Parallel Environment", *EUROGRAPHICS Workshop on Computer Graphics and Visualization*, Pages. 39-52, ISBN: 84-8458-025-3, Girona, Spain, 2001.
- [Zhe07] Ying-Zhe Lue, Yi-Hsing Tseng, "Surface Reconstruction from LiDAR Point Cloud Data with a Surface Growing Algorithm", *Proceedings of the 28th Asia Conference on Remote Sensing*, Kuala Lumpur, Malaysia, 2007.
- [Tamal11] Tamal.K.D, Ramsay.D, L.Wang, "Localized Cocone surface reconstruction", *Computers Graphics*, Volume. 35, Issue. 3, Pages. 483-491, *Shape Modeling International (SMI)*, 2011.
- [Jae05] Jae-Young.S, Sang-Uk.L, Chang-Su.K, "Construction of Regular 3D Point Clouds Using Octree Partitioning and Resampling", *Circuits and Systems. ISCAS 2005. IEEE International Symposium*, Volume. 2, Pages. 956 - 959, 2005.
- [Xiaohui07] Xiaohui Du, Baocai Yin, Dehui Kong, "Adaptive out-of-core simplification of large point clouds", *Multimedia and Expo, 2007 IEEE International Conference*, Pages. 1439-1442, Print ISBN: 1-4244-1016-9, Beijing July 2007.
- [David08] David Belton, "Improving and Extending The Information on Principal Component Analysis for Local Neighborhoods in 3D Point Clouds", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume. XXXVII, Part. B5: 477 ff, Beijing 2008.
- [Yu06] Zhiwen Yu, Hau-san Wong, "An efficient local clustering approach for simplification of 3D point-based computer graphics models", *IEEE International Conference on Multimedia and Expo*, ISBN: 1-4244-0366-7, Toronto, Canada 2006.

Enhancing Human-Robot Interaction by a Robot Face with Facial Expressions and Synchronized Lip Movements

Viktor Seib, Julian Giesen, Dominik Grüntjens, Dietrich Paulus
Active Vision Group, AGAS

Institute for Computational Visualistics
University of Koblenz and Landau, Germany

{vseib, jgiesen, dominik.gruentjens, paulus}@uni-koblenz.de
<http://robots.uni-koblenz.de>

ABSTRACT

With service robots becoming increasingly elaborate for higher level tasks, human-robot interaction is moving into the focus of robotic research. In this paper we present an animated robot face as a convenient way of interacting with robots. Our robot face can show 7 different facial expression, thus providing a robot with the ability to express emotions. This capability is crucial for robots to be accepted as everyday companions in domestic environments. Aiming towards a more realistic interaction experience our robot face moves its lips synchronously to the synthesized speech. In a broad user study with 100 subjects we test the emotions conveyed by the robot face. The results indicate that our robot face enhances human robot interaction by providing the robot with the ability to express emotions. The presented robot face is highly customizable. It is available for ROS and can be used with any robot that integrates ROS in its architecture. Further information is available at <http://ros.org/wiki/agas-ros-pkg>.

Keywords

Robot Face, Talking Head, Animated Dialogue System, Human-Robot Interaction, ROS

1 INTRODUCTION

In recent years robots have found their ways into many homes around the world. As for now, most of these robots are household appliances that were designed to perform one specific task: they are able to vacuum or wipe the floor or to mow the lawn. Nevertheless, the popularity of these, single task specific, robots shows that people are willing to accept robots in their everyday lives.

Therefore, current research focuses on further improving the autonomy and generality of robots. One of the goals in mind are general purpose service robots for domestic tasks. The benefits of having such elaborate helpers at home are manifold. Not only would they take over annoying and tedious household chores, but they could also assist disabled or elderly people in helping them with their daily needs. Especially the last-mentioned aspect is becoming more important in our aging society.

These new application areas require for novel means of communication between man and machine. While it is sufficient to interact with a cleaning robot by pushing buttons on the robot itself or an a remote control, robots strongly integrated in a person's daily routine are expected to understand gestures, speech, and even facial expressions. Likewise, the robot itself has to communicate in a human-like manner using the same means of expressing itself. Since humans focus on faces when communicating with one another, a face also increases the chance of a robot to be accepted as an equal communication partner by a human. A recent psychological study shows that robots exhibiting human-like features are even ascribed more intelligence than robots with less human-likeness [Kra08].

In this paper we present an abstract, cartoon-like, animated robot face for human-robot interaction. While our robot face system possesses only the most important facial features it is able to show 7 essential face expressions that are crucial for human-robot interaction. Additionally, a text-to-speech system is used to synthesize speech by passing arbitrary input strings. The mouth moves according to the synchronized speech and thus provides an even more realistic interaction experience. All animations are generated dynamically during runtime by interpolating between previously defined shape keys. Our animated robot face is available as a package for the widely spread robotics middleware

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

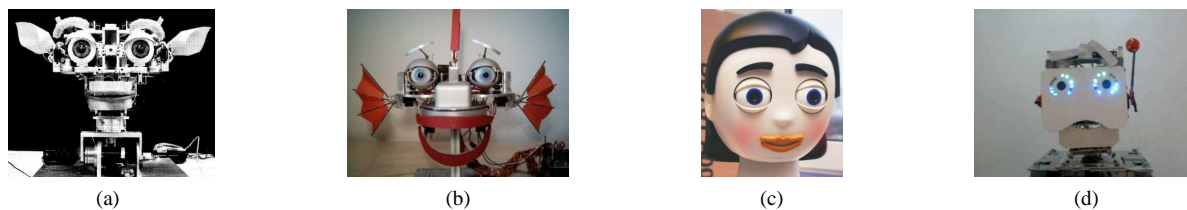


Figure 1: Robot heads designed in hardware: (a) Robot head “Kismet”, Breazeal et al. [Bre99, Bre03], (b) Emotional-display “EDDIE”, Sosnowski et al. [Sos06], (c) Cartoon-like robot head “Flobi”, Lütkebohle et al. [Lüt10], (d) Head of general-purpose social robot “Bender”, Ruiz-del-Solar et al. [Rui09].

ROS [Qui09]. It can be downloaded¹ and easily used on any robot equipped with a display and running a ROS-capable architecture. As it is completely designed in software, the `robot_face` is easily customizable. It is even possible to replace the whole face model by a different one without losing any of the features described in this paper. To our knowledge this is the first easy to use animated robot face that every one can adapt and integrate into an existing robot.

The next Section describes related work and design concepts in some specific aspects that distinguish our animated robot face. The actual implementation is presented in Section 3. Section 4 describes the evaluation procedure of our robot face, followed by a discussion of the results in Section 5. Finally, Section 6 concludes with a summary and an outlook to future work.

2 RELATED WORK AND DESIGN CONCEPTS

Different talking heads were developed in the last years for research in the field of human-robot interaction. Kismet, a robot head demonstrating facial expressions is presented in [Bre99, Bre03]. It expresses emotions by moving its facial features like eyes, mouth and ears. A more recent approach, the emotional-display EDDIE [Sos06], uses the facial action coding system (FACS) [Ekm77] to depict emotions. By defining *action units*, i.e. smallest movable units, FACS describes the movements of most facial muscles and their effect on the face expression. In contrast to these two approaches, Flobi [Lüt10] was designed as a cartoon-like robot head with humanoid features. Its design completely hides the interior mechanics. Another recent approach is Bender [Rui09], which is also able to show emotions. Ruiz-del-Solar et al. conducted a study to evaluate the effect of Bender’s emotion on humans interacting with it. We compare the results of this study with the results of our own study in Chapter 4. The here mentioned robot heads are presented in Figure 1.

The robot heads of these systems are constructed in hardware, posing a challenge in designing and build-

ing these heads. Also, the costs of the different components needed might be an issue. A strong advantage, however, is the possibility to place cameras inside the head’s eyes. This allows for intuitive interaction in a way that a person can show an object to the robot by holding it in front of the robot’s head.

Although this is not possible with a face completely designed in software, we chose this approach to create our animated robot face. In our opinion the high number of advantages of an animated head outweighs its drawbacks. There is no specific hardware that needs to be added to the robot. Thus, there are no additional expenses arising from using our robot face. Moreover, it is highly customizable and can be adjusted to everyone’s individual needs. Finally, the ROS interface allows for comfortable and easy integration in existing systems.

2.1 Cartoon-like Appearance and Abstraction

When focusing on animated faces two main approaches can be distinguished. Human-like or even photorealistic faces are employed to convey realism and authenticity to the interacting person. On the other hand the purpose of stylized cartoon faces is to invoke empathy and emotions. Often this is achieved by exaggerated facial expressions or unrealistic proportions of eyes, mouth or other facial features.

Since our robot (like most of robots participating at the RoboCup@Home) lacks humanoid features and stature, a realistic human face is not appropriate to interact with it. Instead, we modeled an abstract cartoon face exhibiting only the most important facial features to express emotions: eyes, eyebrows, and a mouth. A second reason for the choice of a cartoon face is to avoid the risk of falling into the *uncanny valley*. According to [Mor70], the familiarity of a robot (or a doll, etc.) increases with human likeness. However, when reaching a certain point of high similarity even slight differences from natural appearance cause an uncomfortable effect in the observer. Moving entities augment the similarity with humans, but also the uncomfortable effect. We therefore aimed at creating an animated face that is able to convey familiar face expressions and emotions, but at the same time is not

¹ Package `robot_face` on <http://ros.org/wiki/agas-ros-pkg>

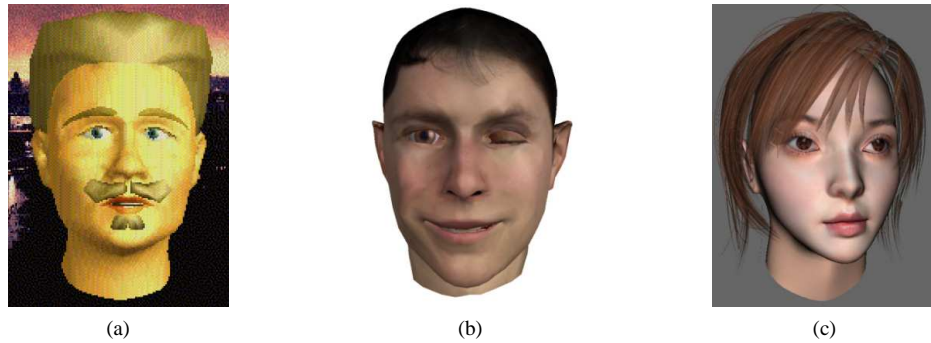


Figure 2: Animated text-to-speech systems: (a) August Dialogue System, Gustafson et al. [Gus99], (b) Facial Animation System, Albrecht et al. [Alb02], (c) Text-to-audio-visual Speech, Niswar et al. [Nis09].

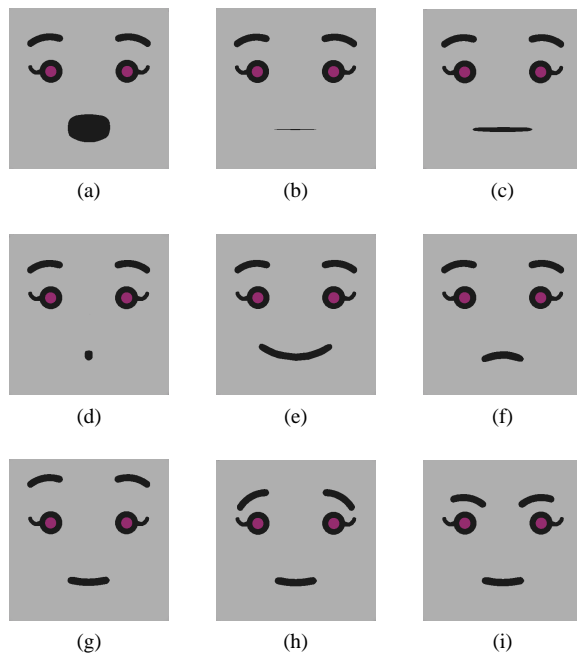


Figure 3: Visemes of our robot face ((a) through (f)) and different shapes of the eyebrows ((g) through (i)).

realistic, i.e. human-like, enough to create an uncanny effect.

2.2 Lip Movement and Speech Synthesis

A key feature of a robot face designed for interaction is the ability to speak. We use a text-to-speech system *Festival*² for speech synthesis. Festival synthesizes speech by applying phonetic and linguistic rules to the input character sequence. To provide an effect of authenticity to the interacting person the lip movements have to be synchronized and animated according to the spoken words of the robot's face. The FACS [Ekm77] is not well suited for this purpose since it does not include the lower face part. We achieve this synchroniza-

tion by mapping visemes to phonemes of the synthesized text. Visemes are visually distinguishable shapes of the mouth and lips that are necessary to produce certain sounds. Phonemes are groups of similar, but not identical sounds that feel alike for the speaker. There are phonemes that produce the same viseme and some that do not alter the shape of the mouth at all. Therefore, only a few visemes are sufficient to achieve a realistic animation of the lips (Figures 3a through 3f). Several animated robot heads were developed in the recent years that possess this skill. Some examples from [Gus99, Alb02, Nis09] and are shown in Figure 2. In contrast to our approach, these animated heads were designed with the goal of modeling a realistic and human-like appearance. To our knowledge none of them was used to interact with a robot.

2.3 Expressing Emotions

Moving the mouth and lips is not enough to allow for comfortable interaction. The movements have to affect the whole face in order to make it appear vivid. A face capable of expressing emotions is crucial for a robot to be accepted as an equivalent communication partner. The face expressions of our robot face are depicted in Figures 5a and 5b.

Animated movies and video games often use animations created manually since the spoken text is known a priori. However, for our purpose only dynamically generated animations came into consideration, as we want to animate arbitrary text with the desired face expression. Apart from visemes we defined shape keys containing several different configurations for the eyes and eyebrows (Figure 3g through 3i).

3 ANIMATED ROBOT FACE

We have developed a talking head application for human-robot interaction named `robot_face`. The talking head performs synchronized lip movements with spoken language and shows 6 different emotions and a neutral face expression. Our goal was to create an application easy to use with robots and to have

² <http://www.cstr.ed.ac.uk/projects/festival/>

the possibility to customize the face. As an example for customization we provide two faces with different genders. In addition, the voice's gender, face color, iris color, and outline colors of the face can be adjusted to the needs of the individual user. With some restrictions, a completely different face can be designed with Blender and used with our application. Please refer to the `robot_face` wiki on the project's website for more information.

To accomplish this application, we used Ogre3d³ as graphics engine for visualisation, Qt⁴ as window manager, and Blender⁵ for creating the Meshes. As mentioned before, Festival is used for speech synthesis and ROS has been chosen to allow for easy integration of our robot face with any robot using ROS.

3.1 Face Modelling and Animation

We designed two similar, cartoon-like faces (a male and a female one) for the presented robot face. Both faces were designed with Blender including a mouth for speaking, eyes for blinking, and eyebrows to intensify emotions. The difference between both faces are distinctive eyelashes on the female face and thicker eyebrows on the male face, as well as a different eye color.

Since we modelled our faces with Blender we used polygon models and adapted them with subdivision surface methods. According to [Par02] subdivision surfaces are a good modelling type for cartoon-like faces. We used the modelling method introduced by Jason Osipa [Osi03], where the model is created by hand and which is an excellent way to model a cartoonish face. According to this method, the mouth and eye areas are modelled separately and are connected afterwards. As we need a mouth for automatically generated animations, we modelled it slightly different than described by Osipa. Focus was put on animation during the modelling process. Thus, we created shape keys for all different face movements and emotions. An overview is given in Figure 3.

For mouth movements we limited the number to the four most important visemes namely mouth open, closed, wide, and narrow. With those four visemes, it is possible to create two clearly separated speech cycles: open and close movements together with wide and narrow movements. It is not necessary that both speech cycles are executed at the same time nor do they have to blend from one extreme into the other [Osi03].

Open and close movements occur by almost any sound as opposed to wide and narrow movements which are associated with the art of sound. There are about 38 to 45 phonemes in the English language, but only a few

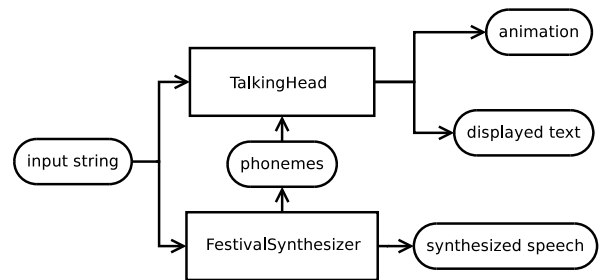


Figure 4: Components and interaction of the robot face.

visual counterparts. Thus, we combined the indistinguishable phonemes into one appropriate viseme.

Beside those visemes we have designed other mouth shape keys for emotional representation. We used 6 different emotions namely happy, sad, angry, surprised, scared, disgusted, and also a neutral expression. These emotions are shown in Figures 5a and 5b. Additionally, we added shape keys for eyebrows (up, middle up, middle down). These are shown in Figure 3. To achieve movement, the shape keys are interpolated in our developed application with the use of Ogre3D.

3.2 Structure of `robot_face`

Our `robot_face` application consists of two ROS-nodes. The `TalkingHead` node manages both the mesh and the animation. To get even better feedback on what the robot says it also displays the spoken text under the robot face. Furthermore, emoticons that are used to specify the robot's face expression are removed from the displayed text. The creation of phonetic features including speech and voice is handled by the `FestivalSynthesizer` node. An overview is given in Figure 4.

We use the messaging system of ROS to communicate with `robot_face`. In order to do this, a string needs to be published on a specific ROS topic. It is directly delivered to the application where it gets synthesized, animated, as well as displayed. In detail, if a given text is sent via the message system to `robot_face` it arrives at the two ROS-nodes `TalkingHead` and `FestivalSynthesizer`. The `TalkingHead` displays the text for the duration of the animation. It is also capable of displaying additional information (i.e. robot state, recognized speech) published as string to a different topic.

`FestivalSynthesizer` synthesizes the speech. It generates phonemes and speech corresponding to the provided text using Festival. We use PulseAudio⁶ as sound system for audio output. Apart from the phonemes corresponding timestamps are generated by the `FestivalSynthesizer` node. This information is used by the node `TalkingHead` for animation.

³ <http://www.ogre3d.org/>

⁴ <http://qt.nokia.com/>

⁵ <http://www.blender.org/>

⁶ <http://www.pulseaudio.org/>

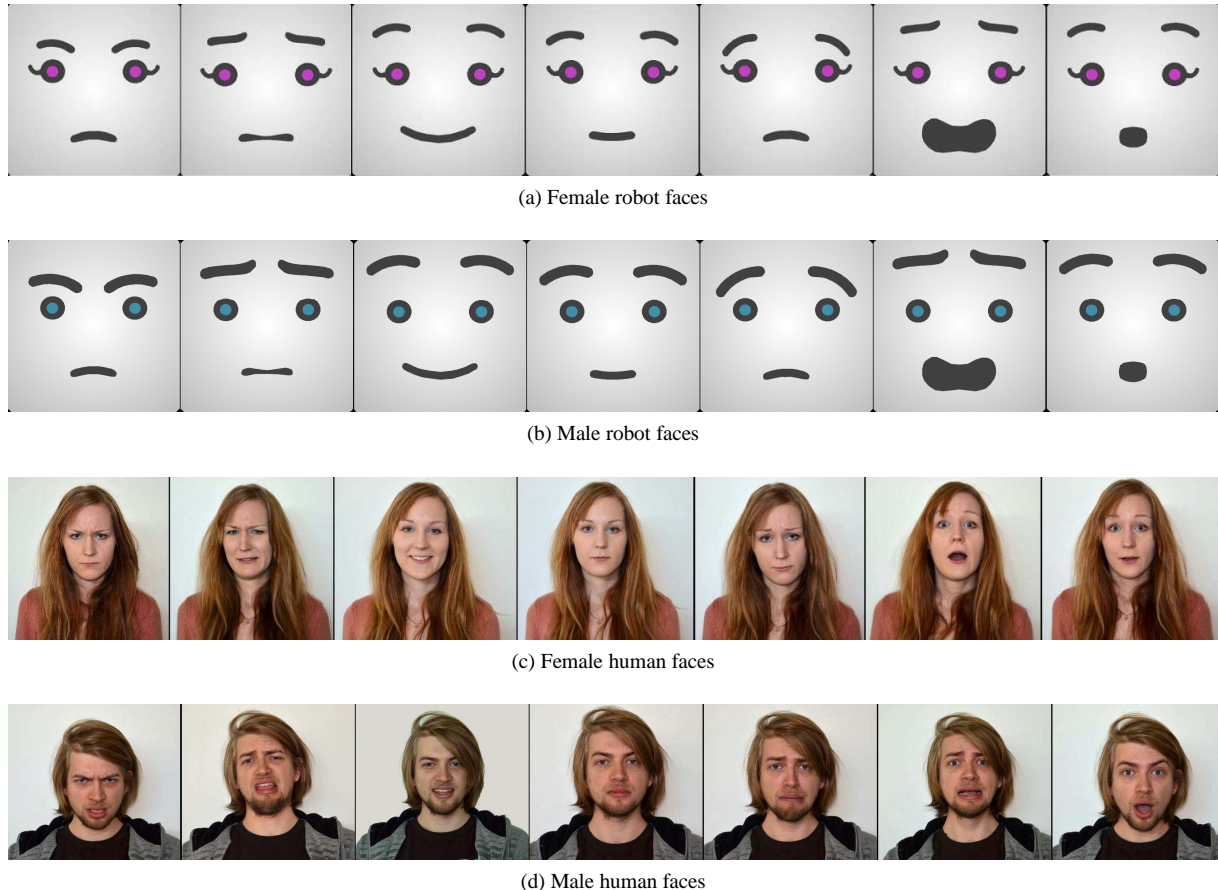


Figure 5: Face expressions that can be displayed by our robot face and the corresponding face expressions of our human models for evaluation (from left to right): angry, disgusted, happy, neutral, sad, frightened, and surprised.

In the TalkingHead node the face mesh is animated by Ogre3D. The main structure of TalkingHead is organized into the creation of the scene, creation of animation, and play-back of animation.

The submeshes of the loaded mesh are counted and the same number of animations is created. These animations need to get filled with keyframes to represent movement. By default, incidental blinking and wiggle animations are active. Keyframes are generated with the phonemes and timestamps mentioned before. We build a predefined phoneme-viseme-map to associate phonemes with visemes. A keyframe is generated for every viseme and emotion using the timestamps. The keyframes are then connected to a whole animation. As soon as the animation starts the spoken text is displayed below the robot's face.

4 EVALUATION

Similar to the evaluation presented in [Rui09], we evaluated the presented robot face to determine how the intended face expressions are perceived by people and whether the intended emotions could be conveyed. Further, we tested how comfortable people were when looking at the developed robot face. The results of both evaluations are compared and discussed in Section 5.

The evaluation was performed as an online questionnaire. The test was divided into two parts, each having 14 questions. In the first part the test persons were presented all 7 face expression of our robot face (Figure 5a) and a photo of a human face expressing one of these emotions (one of the photos in Figure 5c). The probands had to select the robot face that best matched the face expression of the human. Although the presented human face always was intended to show one of the displayed robot faces, the test subjects also had the possibility to select *unknown* and thus skip the question if they could not decide. This test was performed once for each of the 7 face expression in Figure 5c, each time with a different photo. Subsequently, all 7 questions were repeated in a different order with a robot face depicting a male face (Figure 5b) and photos of a male human (one of the photos in Figure 5d). In this part of the test no adjectives describing or naming any of the face expressions were involved.

In the second part the probands were presented one of the robot faces and had to select from a list with 14 adjectives which described the displayed face best. The 14 adjectives contained the 7 available expressions, 6 expressions that were not depicted by the robot face,

Table 1: Results of the first part of the evaluation. Each line represents a photo of a human face with the indicated expression. The numbers show which robot faces were matched to the displayed photo (in percent). Matches above 10 % are printed in bold, the maximum of each line is marked gray.

	matched with	angry	disgusted	happy	neutral	sad	frightened	surprised	unknown
angry	85	8	0	0.5	0.5	1	0	5	
disgusted	6.5	34.5	0	1	7.5	26.5	2	22	
happy	0.5	0	76.5	19	0	1	0	3	
neutral	4.5	3	0.5	87.5	2.5	0	0.5	1.5	
sad	1	75	0.5	1.5	21	0.5	0	0.5	
frightened	0.5	5.5	0	0	3.5	76	12.5	2	
surprised	0	2	0.5	5	1.5	12	77	2	

Table 2: Results of the second part of the evaluation split in two halves. The upper half contains presented face expressions, while the lower part contains face expressions that were not shown to the test subjects. Each line represents the robot face with the indicated expression. The numbers show which expression was matched to the displayed robot face (in percent). Matches above 10 % are printed in bold, the maximum of each expression assigned is marked gray.

	identified as	angry	disgusted	happy	neutral	sad	frightened	surprised
angry	81.5	0.5	0	0	8	0	0.5	
disgusted	2	1.5	0	0	19.5	1	0	
happy	0	0	94	2.5	0	0	0.5	
neutral	0	0	3.5	88.5	1	0	0.5	
sad	0.5	0	0	0	87.5	0.5	0	
frightened	3	8	0.5	0	0	70.5	6	
surprised	0	0	0	0	0	4.5	90.5	

	identified as	anxious	tired	bashful	bored	arrogant	hurt	none of these
angry	0	0	0.5	1.5	1	6	0.5	
disgusted	39.5	1	12	1	0.5	19.5	2.5	
happy	0	0	0	0	0	0	3	
neutral	0.5	1	0.5	2	0	0	2.5	
sad	2	2	1.5	1.5	0	4.5	0	
frightened	7	0.5	0.5	0	0.5	1	2.5	
surprised	0.5	0	0	0	0	4	0.5	

and the option *none of these*. Again, this was tested for each of the 7 robot face expressions, first with female then with male robot faces.

A total of 100 persons (62 male, 38 female) aged between 19 and 58 years (average 26.2 years) participated in our evaluation. To 53 persons the face of our robot was unknown before the evaluation. 34 people stated to have seen the face before, but to have never interacted

with the robot. The remaining 13 persons knew the face and also had interacted with the robot.

5 RESULTS AND DISCUSSION

Each part of the evaluation was performed with male and female faces (either human or robot). The results of both genders were averaged for each part of the evaluation and are presented in Table 1 for the first part and

Table 3: Comfort of the test subjects when looking at the robot faces presented in the evaluation.

very uncomfortable	uncomfortable	undecided	comfortable	very comfortable
2 %	5 %	38 %	46 %	9 %

in Table 2 for the second part. Each line in Table 1 represents a photo of a human face showing the face expression indicated in the first column. Accordingly, every line in Table 2 stands for a robot face with the given expression. The numbers are percentage values and indicate which robot faces were matched to the displayed photo (Table 1) or the expression the robot face was identified as (Table 2). Every case above 10 % is printed in bold, the maximum of each line is marked gray. Ideally, the diagonal would show 100 % at each position in Table 1 and in the first half of Table 2.

Most of the elements in the diagonal of Table 1 have high values: 5 have values of over 75 % and 2 of them have 85 % or more. Only 2 of 7 photos were not matched well with the provided robot face expressions. This is a strong indication for the fact that the key facial features of our robot face are able to recreate the face expressions of humans correctly. The misclassifications in the first part can also result from misclassification of the presented human face. Thus, in the second part of the evaluation no human faces were presented to the probands. The diagonal of Table 2 has 6 elements with more than 70 %, 3 of these have more than 80 %, and the other 2 even over 90 % identification rates. When the robot faces are evaluated on their own without being compared to human faces, only 1 of 7 does not match the intended expression.

In Table 1 the expressions *angry* and *neutral* have the best matches and were not falsely related to other robot faces (i.e. no other columns with 10 % or above). Table 2 confirms this findings. Thus, these two face expressions can be classified well on their own and even pass the comparison with a human photograph.

The *happy* photo was matched correctly with the corresponding robot face in 76.5 % of cases. However, almost every fifth proband assigned the neutral robot face to this photo. Comparing this result to Table 2 shows on the other hand that the *happy* robot face has the highest correct classification result of 94 %. Thus, the high misclassification rate when directly compared to a human photo stems from the human face expression and not from the robot face.

A look at the expressions *frightened* and *surprised* shows a duality in Table 1. Both have very similar correct matches, but were at the same time misclassified with one another - again with very similar rates. Table 2 shows again that this error must result from the human face expression on the photo since the robot faces were misclassified with a significantly lower rate.

The expressions *disgusted* and *sad* have bad matching results in the first part of the evaluation. When presented on its own, the *sad* robot face has excellent classification results (Table 2). However, the *sad* human photo was mostly matched with the robot face that shows a *disgusted* face. Thus, while the *sad* robot face is indeed perceived as sad the *disgusted* robot face seems to resemble better the features of sad human faces. On the other hand, the *disgusted* photo was matched to the correct robot face in only 34.5 %. Over one fourth of all test subjects matched it with the *frightened* robot face. Further, the high number of probands that selected *unknown* indicate that non of our robot face expressions can resemble the features of disgusted human faces. This findings are confirmed by the results in Table 2 where almost no correct identifications for the *disgusted* face are present (only 1.5 %). The *disgusted* robot face was mostly classified as *anxious* (39.5 %), *sad* (19.5 %), *hurt* (19.5 %) or *bashful* (12 %). The various maxima in the classification of this robot face show that it is difficult to identify and to be assigned a feeling to. However, considering that sad and hurt are similar expressions, it can be stated that the *disgusted* robot face resembles an anxious or a sad face expression.

In contrast to Bender [Rui09], who can show 4 different face expressions, our robot face can show 7. Compared to the results of the evaluation of Bender, our robot face achieves higher recognition rates by the test subjects. The highest difference occurs with the *happy* face expression, where our application was recognized correctly in 94 % of cases (compared to 51 % of Bender). The other 3 face expressions compare as follow (results for Bender given in brackets): surprised 90.5 % (76.5 %), sad 87.5 % (78.4 %), and angry 81.5 % (76.5 %). One needs to take into account that Bender is a hardware robot head and looks more technically compared to our cartoonish animated robot face. It is obvious that designing a robot head in hardware with several facial expressions is more challenging than in software.

Apart from the classification of the presented face expressions the test subjects were asked to rate their comfort when looking at the robot's faces. The results are shown in Table 3. While only 7 % of the probands experience discomfort, 55 % feel comfortable when looking at the presented robot face. Although, the number of undecided test subjects is high the results indicate that our robot face does not fall into the *uncanny valley*.

6 CONCLUSION AND FUTURE WORK

We presented an animated robot face that is able to show 7 different face expressions and whose lips are synchronized to the synthesized speech. This robot face is highly customizable and can be used with any robot running ROS.

An evaluation with 100 test subjects shows that 5 of 7 robot faces were correctly assigned to a presented human face in 80 % (average) of all cases. Also, 6 of 7 robot face expressions are classified correctly in 85 % on average. This is a strong indication that our robot face enhances human robot interaction by providing the robot with the ability to express emotions. Compared to a similar evaluation of a state-of-the-art robot face in hardware, the presented approach performs significantly better in a user study.

The only face expression not classified correctly by most users was the face expression that we intended to show disgust. According to the results of the user study this expression conveys a mixture of anxiety and sadness and thus should be used accordingly.

The evaluation also shows that most probands (55 %) feel comfortable when looking at the robot face, while 38 % are undecided. This and the reason that it is a cartoon face leads to the assumption that it does not fall into the uncanny valley, although more investigation in this area is desirable.

Our future work will concentrate on improving the ability of our robot face to express emotions. For instance, the appearance of the robot's eyes can be changed depending on the presented emotion. Also, a new face expression for disgust needs to be found as the current one will be used as anxiety and sadness in the future. Further, with the fact in mind that our robot face can express emotions as is shown by the presented evaluation, we want to evaluate whether it can invoke empathy in humans interacting with a robot that is equipped with the presented robot face.

7 ACKNOWLEDGEMENTS

The authors would like to thank Alruna Veith and Lubosz Sarnecki for posing as face models for the evaluation photos.

8 REFERENCES

- [Alb02] Albrecht, I., Haber, J., Kahler, K., Schroder, M., and Seidel, H.P.; May I talk to you?:-)-facial animation from text. In *Computer Graphics and Applications*, 2002. Proceedings. 10th Pacific Conference on, pages 77-86. IEEE, 2002.
- [Bre03] Breazeal, C.; Toward sociable robots. *Robotics and Autonomous Systems*, 42(3):167-175, 2003.
- [Bre99] Breazeal, C. and Scassellati, B.; How to build robots that make friends and influence people. In *Intelligent Robots and Systems*, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on, volume 2, pages 858-863. IEEE, 1999.
- [Ekman77] Ekman, P. and Friesen, W.V.; Facial action coding system. 1977.
- [Gus99] Gustafson, J., Lundeberg, M., and Liljen-crants, J.; Experiences from the development of August - a multi-modal spoken dialogue system. In *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems (IDS-99)*, 1999.
- [Kra08] Krach, S., Hegel, F., Wrede, B., Sagerer, G., Binkofski, F., and Kircher, T.; Can Machines Think? Interaction and Perspective Taking with Robots Investigated via fMRI. *PLoS ONE*, 3(7), 2008.
- [Lüt10] Lütkebohle, I., Hegel, F., Schulz, S., Hackel, M., Wrede, B., Wachsmuth, S., and Sagerer, G.; The Bielefeld Anthropomorphic Robot Head Flobi. In *2010 IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, 5 2010. IEEE, IEEE.
- [Mor70] Mori, M.; Bukimi no tani [The uncanny valley]. 1970.
- [Nis09] Niswar, A. and Ong, E.P. and Nguyen, H.T. and Huang, Z.; Real-time 3D talking head from a synthetic viseme dataset. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 29-33. ACM, 2009.
- [Osi03] Osipa, J.; Stop Staring - Facial Modeling and Animation Done Right™. Sybex, 2003.
- [Par02] Parent R.; Computer Animation - Algorithm and Techniques. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Academic Press, 2002.
- [Qui09] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Ng, A.; Ros: an open- source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [Rui09] Ruiz-del-Solar, J., Mascaró, M., Correa, M., Bernuy, F., Riquelme, R., and Verschae, R.; Analyzing the human-robot interaction abilities of a general-purpose social robot in different naturalistic environments. In *Lecture Notes in Computer Science (RoboCup Symposium 2009)*, volume 5949 of LNCS, 2009.
- [Sos06] Sosnowski, S., Bittermann, A., Kuhnlenz, K. and Buss, M.; Design and evaluation of emotion-display eddie. In *Intelligent Robots and Systems*, 2006 IEEE/RSJ Int. Conf. on, 2006.

Interacting in 3D Virtual Worlds with Brain Computer Interfaces

Janek Ilgner Robin Kuhlmann Helmut Eirund Martin Hering-Bertram

Hochschule Bremen University of Applied Sciences
Flughafenallee 10
D-28199 Bremen
www.hs-bremen.de

ABSTRACT

Interaction with 3D virtual worlds found in first-person action games is mainly based on keyboard input or pointing devices. Console games add new input devices like motion capturing or voice control. Though immersion is a key issue, most games do not rely on player's emotions. To take this important communication factor into account, we propose a method capturing the player's emotions of anxiety and shock in a game and use this data to control player's and non-player-character's actions. We present a game setup that is specifically designed to evaluate the use of emotional interaction based on a small user study. A simple EEG based Brain Computer Interface (BCI) is used to translate amplitudes of alpha (stress) and beta (shock) rhythms to corresponding commands in the game engine. The game is set in a horror scenario in which the player needs to control his emotions as they may adversely influence the difficulty of the gaming tasks. The game concept implements an immersive atmosphere to bind the player emotionally and evoke signals captured by the BCI. The game engine passes these emotional inputs to actions of game entities (visuals and opponent's reactions). Finally, the impact of emotional interaction is evaluated by a small group of test players projecting the needs for future approaches and enhancements.

Keywords

Brain Computer Interfaces, BCI, EEG, game design, emotions, immersion, multimodality, evaluation

1. INTRODUCTION

With the release of Nintendos Wii and its gesture recognition technique to control movements, a lot of new input modalities have been developed. No one would doubt that these new modalities took a major role in the success of these systems. Most of these modalities, however, focus on gesture or voice recognition. Very little attention is put to using the player's emotion as additional gameplay element.

A well designed, immersive, game can evoke emotions for the user. This communication between

the game and the player, however, is limited to one direction since the game does not have the means to capture emotional reactions to the situations it creates.

Bidirectional communication, however, would enhance not only the gaming experience, but also alleviate human-computer-interaction. In human-to-human interaction the emotional level plays a very important role to decipher the meaning of the information conveyed. An emotional input modality would close this gap in human-computer communication. For interactive games this would possibly result in a higher immersion, since the communication with the game would feel more natural and the additional modality would draw the players focus even more to the virtual world.

To approach the issue of emotional feedback, we have created an immersive computer game, located in the horror genre, based on an EEG based Brain-Computer-Interface (BCI) to capture the emotions of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

shock and anxiety/stress and translate them to actions and events in the game.

Usually the emotions have a negative effect for the player, hindering his process, so he needs to monitor his emotions constantly and thereby is focused on his emotional reactions to the events in the game.

Since a game can only evoke emotions if the player is immersed in it, we analyzed and implemented several factors creating immersion and supporting the emotional binding of the player in the game.

Contributions

Our brain-computer-based game development and its evaluation is directed at the following goals

- we propose multiple interaction patterns where the user's emotions directly influence his perception and the action of his opponents
- a game concept using these patterns and their key implementation issues are provided
- based on a small user study, we assess the possibilities and limitations of brain-computer interaction in games

Though our work is highly experimental and our user study is far from being representative, we are able to provide a proof of concept showing the full benefit of emotional sensing, anticipating its future advances in game development.

Overview

The remainder of our work contains an overview of EEG-based Human-Computer Interaction (HCI) in section 2. Section 3 introduces our game design and shows how the emotional user input can be processed to enhance immersion. Sections 4 and 5 contain implementation issues and the results we obtain from our experimental user study, respectively.

2. BACKGROUND

Human-Computer Interaction

A Brain-Computer-Interface is a communication system that allows the user to communicate to a machine or the surroundings, without relying on signals from peripheral nerves or muscles [Nic12a].

Since the beginnings of BCI development in the 1970s for military purposes, the focus of research

increasingly concentrated on medical and, more recently, on entertainment uses.

Today, there is a rising interest in BCI research, mainly due to more powerful and affordable hardware, but also because of a rising public interest and acceptance of BCI use to aid disabled people and successful studies in this field. The number of active research groups on BCI related topics went up from six to eight 10 years ago to currently over a hundred [Nic12a].

Many gamers are early adopters and open to new technologies. They are also used to invest time to learn and master a game. Competition is also a big part of gaming and new communication or movement modalities could be a benefit. That makes games interesting for BCI research and even researchers for medical applications have looked at games to find training solutions for patients. [Nij09a]

Brain-Computer-Interfaces are usually divided in two classes: Dependent and independent [Wol02a]. While an independent BCI completely ignores common output pathways of the brain and offers new communication channels to the user, a dependent BCI still relies on them to some extent (e.g. a BCI may depend on electrical potential differences generated by visual evoked potentials).

Electroencephalography (EEG)

Most modern Brain-Computer-Interfaces depend on EEG for signal acquisition. EEG based BCIs offer a relatively easy to set up and risk free way of recording brain activity.

Signals are acquired by placing electrodes on the users scalp that measure electrical activity generated by ionic currents flowing within and across neurons, see figure 1. Although the latency is quite small the signal quality is often poor and the system is easily distorted by background noises either in the brain itself or from external sources [Nic12a].

Due to the difficulties with EEG-based BCIs, users need to invest time to learn how to make the desired inputs. However, during a study executed in 2004 at the Fraunhofer Institute they managed to minimize the training time through the use of neural networks. Parts of the learning process were now transferred to the computer and successful results were possible in about 30 minutes of training. [Car06a].



Figure 1: EEG-based sensing, http://www.emg.tu-bs.de/bilder/forschung/eegekg/eeg_w.jpg

Nonetheless, certain signal patterns recorded by EEG seem to be connected to specific mental activities. Basically the acquired signals can be divided in two classes, evoked or event related potentials and EEG rhythms. Evoked potentials are potential differences recorded on a limited area of the brain. They reflect physical (evoked potentials) or mental (event related potentials) stimuli.

If the mental activity of a BCI user is recorded on a large scale with EEG, certain rhythmic patterns emerge that can be classified into different EEG rhythms. These rhythms can be linked to certain brain functions, since they desynchronize with specific mental tasks or activities carried out. This behavior is called event-related desynchronization (ERD) and can help to interpret signals recorded in the EEG [Pfu99a].

Name	Frequency	Description
Alpha (α)	8-13 Hz	Mental or visual effort
Beta (β)	13-30 Hz	Motor activity

Table 1: Frequency range of alpha and beta signals.

For the emotion aware game that has been developed, two of these rhythms, listed in table 1, were particularly interesting. One of them is the alpha rhythm. The alpha rhythm is continuously developing over the first ten years of a human. At the age of ten the mean alpha frequency of adulthood is reached. It attenuates or suppresses at a degree of higher alertness [Nie99a] and can be linked to mental or

visual effort and is used in the game to keep track of the players stress level. When the player is at a calm, relaxed state the alpha rhythm does not desynchronize, while at a stressful situation this changes with increasing mental activity or rapid eye movement.

The second rhythm used is the beta rhythm which can be linked to motor activity or tactile stimulation of the BCI user, even if it is just mentally imagined motor activity [Nie99a]. This is used to catch the shock emotion of the player. If the player is shocked he will most probably shudder or wince thereby using his muscles which then serves as an indicator for the system to trigger related actions in the game.

EEG-based BCI devices have been used for games before. Besides the NIA Game Controller, described in Section 4, there are devices like the “Mindset” (NeuroSky Inc. 2009) and the “Epoc” (Emotiv Inc. 2009) that control a variety of applications. “NeuroBoy” for example is a game in which the player has to focus or relax to achieve certain goals while in “Stonehenge” motor movements are used to reassemble fallen pieces of the Stonehenge [Tan10a].

Engagement	Engrossment	Total Immersion
<ul style="list-style-type: none"> - Become focused - Lose track of time 	<ul style="list-style-type: none"> - Emotions are directly affected - Wants to keep playing - Game becomes most important part of attention - Less aware of surrounding / less self aware 	<ul style="list-style-type: none"> - Cut off from reality / Game is all that matters - Feels attached to a main character or team
Access: Player's preferences, game controls Investment: Time, effort, attention	Game construction: Combination of Game features	Empathy: Growth of attachment Atmosphere: Development of game construction

Effect on player	Barriers
------------------	----------

Table2: Levels of Immersion [Bro04a]

Immersion in Computer Games

For a game processing shock emotions, it is important to create a virtual world in which the

players can immerse. When players are immersed, their emotions are directly affected by the game [Bro04a] and emotional reactions happen more frequently. Brown and Cairns divided Immersion into three levels and described which barriers must be lowered to get to them, see table 2.

One part of the game construction is storytelling. Mateas [Mat00a] integrated the concept of agency in Aristotle's theory of drama. Agency is a feeling of control and empowerment that players can get when their actions in the game world relate to their intentions. The players will experience agency when the material for action is balanced with the affordances of the game world. For example, if the game suggests that the player can pick up an object but it's not possible to do so, the sense of player agency will decrease.

Roth et al. [Rot09a]. described experiential dimensions that can motivate players:

- Curiosity (“What will happen next?”),
- suspense (“Will they survive?”),
- aesthetic pleasantness (“Beautiful!”),
- self-enhancement (“We are great!”) and
- optimal task engagement (“Don’t disturb me!”)

3. GAMING CONCEPT

Goal

The goal of our work is to create a game implementing a Brain Computer Interface as an additional modality to identify and process emotions of the players. The game is set in the horror genre because it is suitable for creating mental stress and shock moments which can then be captured by EEG. To create such situations, methods facilitating immersion had to be adopted.

Preparation

Before implementing the game the required components had to be analyzed and evaluated. This includes a Brain-Computer Interface that is capable of analyzing the relevant parameters and carrying out corresponding configurable actions.

On the other hand, a game engine is needed that is able to interpret the actions coming from the BCI. Also, this engine needs to be technically able to create an immersive feature-rich atmosphere.

For fulfilling these requirements we decided to use the NIA Brain Computer Interface by OCZ Technologies and the Unreal Development Kit for the Unreal Engine 3 by Epic Games Inc.

To test the functionality of the BCI with sensing emotions in horror games some tests were carried out with a test subject playing a horror game (Amnesia:

The Dark Descent by Frictional Games). The test included three different camera views: The game itself, the parameters of the NIA BCI and the face of the subject.

By doing this, a correlation between the muscular face movements and the beta rhythms of the EEG could be observed. Also the anxiety seemed to strongly correlate to the stress the subject was experiencing.

To evaluate the possibility to connect the NIA output to the game engine the NIA user interface was found to be able to output key presses as actions when certain amplitude thresholds were met. So the UDK only had to take these inputs and translate them to corresponding actions.

Game Concept

To create an atmosphere evoking mental stress and exposing the user to shock moments, the game takes place in a dark forest inhabited by strange creatures, plants and objects. The players have to walk across this forest after they crash with a hang glider in the mountains. The players have no weapons thereby the only option during most of the enemy confrontations is to flee. The BCI captures mental stress and shock reactions of the players. The game gets more difficult when they are stressed or shocked. For example, enemy tendrils in the forest grow and spikes shoot through the ground when the players can't keep calm.

Creating Immersion

To create an immersive game world the barriers had to be lowered so the players can get to the level of total immersion.

To lower the access barrier the game's controls are similar to the controls of a first-person shooter. This assures that everyone who played a first-person shooter before is familiar with the control scheme.

To lower the investment barrier the players have to invest time effort and attention. To give them a motivation to do so the game starts with an introduction in which they learn something about the initial situation of the story. This is to make the players curious and motivate them to find out what's going on.

The most complex barrier that has to be lowered is the game construction because it consists of many game features like visuals, sound, plot and challenges.

The research showed that most of the games players felt immersed in were played in first-person perspective. This also reduces the risk that a player cannot identify with a predetermined character.



Figure 2: Gate is blocked by tendrils



Figure 3: Dark atmosphere

The game's story is told by the game world and its creatures and objects. To keep players motivated the creatures and objects are introduced one after another following an arc of suspense that lead to a climax which is a boss battle. Constructions and altars suggest the presence of an ancient, friendly civilization and the tendrils and enemies stand for an evil infestation of the forest, as illustrated in figures 2 and 3.

To create the feeling of agency the game tries to avoid situations in which the player's actions have no meaning for the plot. The number of possibilities for interaction is always at a manageable level and every object for interaction must be used to proceed with the game. For example, there is a situation in which a gate is blocked by tendrils. Beneath the gate is an altar with small, gray mushrooms on it. Nearby this gate players can find a blue mushroom which is the only object to interact with. When the mushroom is put on the altar, a cutscene shows how the mushroom is growing and the tendrils are moving back and unblock the gate. This scene teaches the players that the mushrooms and altars are helpful while the tendrils hinder them to move forward. We hereby also address the experiential dimensions described by Roth et al:

- Curiosity: "What are these tendrils?"
- Aesthetic Pleasantness: "The way the mushroom grows and the tendrils move back looks very nice."
- Self-Enhancement: "My idea with the mushroom worked right away. I am clever!"

Music and sound effects have a big impact on the atmosphere of a game. To create music that fits a mysterious and eerie forest, the soundtrack uses minor chords and deep bassy sounds.

Furthermore, the speed of the track has to collaborate with the situation in the game. Therefore the game has slow ambient music with emphasis on bass when the players explore the forest and hectic music with a fast drum beat when they get chased by enemies.

Atmospheric sounds describe the surroundings and actions of the players. For example, in the forest they can hear the rustle of the wind in the trees, animal calls and the sounds of their own footsteps. Some sounds are related to certain objects like altars and enemies. There are also special hint sounds playing when events occur in the presence of the player.

Obviously, the visuals have a great influence on the game. To create an atmosphere that supports mental stress and allows shock moments the forest consists of closely spaced trees from which the players often can't see more than the silhouettes. Because of a dark twilight and dense fog the surroundings are uncovered little by little and the paths are bordered by tendrils that are moving slowly. This is to give the players the feeling that something lurks around in the forest that can savage them at any time.

Sometimes the dark atmosphere is intercepted by peaceful places which are brighter and more colorful to avoid that the players get used to the dark and thereby lose their anxiety. This is also a way to arouse their curiosity. Furthermore the changes of mood are important to make the effect of the BCI appreciable.

To lower the barrier atmosphere, the objects in the game (see figure 4) need to be relevant for the player. To ensure this, all the objects have a meaning. Plants with red berries help to navigate through the forest, blue mushrooms work as a source of energy, and altars give hints. Tendrils (figure 5) block the way to places where the players are not supposed to go or hurt them. The different tendrils have a similar appearance and at the end the boss can be identified as their origin.



Figure 4: Objects in Mori: Altar, Plant with red berries, mushroom



Figure 5: Different tendrils

Enhancement of the game experience due to player's emotions

To enhance the player experience through the use of his emotions meaningful, the game implements multiple components that react to them. The plants in the game are introduced early to the player as emotionally sensitive. They hinder his process the more he shows fear and shock. Is the player close to a tendril it begins to glow red, grow and becomes more aggressive thus making it harder for the player to pass. Also thorns can appear almost everywhere in the game world, which hurt the player on contact.

Of course, the player is dragged deliberately in stressful and terrifying situations, especially when near an area of tendrils. There is, for example, a situation where he is being followed by enemies, who want to attack the player. He then has to run through a passage covered in tendrils to escape these enemies.

Also, the final enemy, who is presented as the root of all tendrils, is aware to the shocked state of the player and starts his attacks when he senses this emotion. To shock the player, there are thorns constantly emerging from the ground if he is near them.

The second analyzed emotion is the stress level of the player. A higher stress level results in a manipulated view for the player. The field of view broadens and a material is overlaid resulting in a tunnel view that makes it more difficult more the player to navigate through the world. When he stays calm for some moments, the view returns to normal so he has to try to stay calm especially in stressful situations to be able to pass through them more easily.

After the player gets used to the emotional interaction, she actively has to control this interaction

method in both directions: there is also a situation where the player has to deliberately fake his emotions to proceed. It consists of a trap where a tendril attacks an area on top of a bridge as soon as it senses shock. The player needs to cross that bridge two times. While the first time he has to stay calm to prevent himself from being hurt, the second time there is an enemy on the bridge, blocking the path. To get rid of this enemy, the player needs to fake his shock emotion so that the enemy gets attacked by the tendril.

This experience makes it possible for the player to get an insight of the way the Brain-Computer-Interface works so that he is able to estimate the reactions of game objects to his emotions. By this he is able, for example, to fake his emotions in the fight against the final enemy, timing the attacks of the enemy so that he can get pass it safely.

4. IMPLEMENTATION

NIA Game Controller

The NIA (Neural Impulse Actuator, figure 6) is an EEG-based BCI capable to capture alpha and beta rhythms. It comes with an easy-to-use user interface for configuring input triggering behavior when certain thresholds in the amplitudes are reached.



Figure 6: The NIA BCI

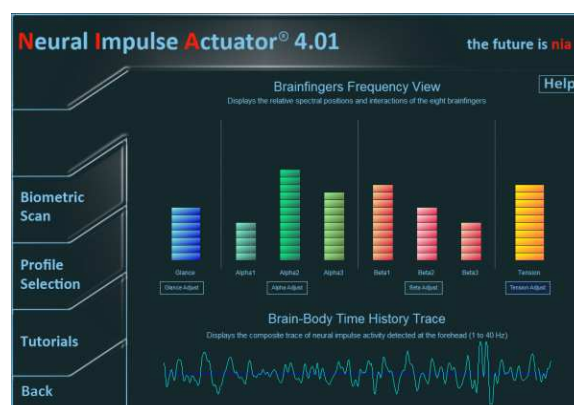


Figure 7: Brainfingers, the software to visualize the BCI parameters

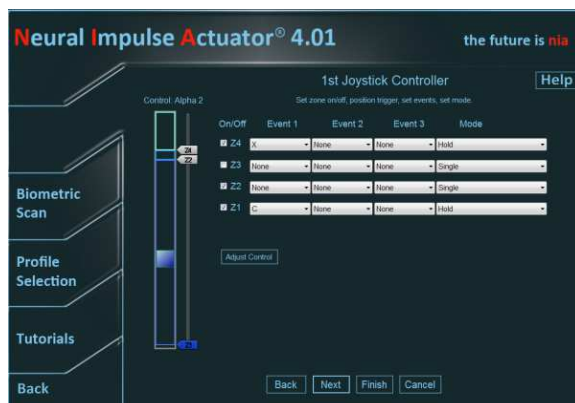


Figure 8: Configuration of parameter thresholds

When the amplitude reaches a defined threshold defined in the NIA software interface (figures 7 and 8), external input controls are addressed, e.g. holding down or pushing a button on the keyboard. This can be used to forward EEG input to other applications, like the game created.

Using the NIA Game Controller to capture the shock and stress/anxiety emotion

To capture the shock or stress emotion, certain thresholds for the corresponding EEG rhythms must be defined in the NIA interface.

When the player is in a completely relaxed state, alpha and beta amplitudes are quite low and should not trigger any actions.

As the stress emotion can be linked to the alpha rhythm, as described formerly, there are two triggers defined: One covering the upper section of the effective range and one at the very lowest. So the alpha amplitude strength rises if the mental state of the player is active and/or a lot of visual processing is taking place (for example by rapid eye movement). When the alpha amplitude reaches high levels, a trigger is activated that presses (and holds down) a predefined key later used in the game to notify the game engine that the stress level is rising.

The same is done with the stress lowering trigger and the shock emotion for the beta rhythm (except this only triggers a single key, since the shock emotion is a one-at-a-time event).

The game engine on the other hand takes this key input and translates it to certain actions that manipulate game objects or the user view. A tendril, or thorns, that suddenly appear may shock the player, so he will shudder or wince, which will consequently make the beta amplitude rise and activate the corresponding actions. Alpha and beta signals for different states are depicted in figure 9.



Figure 9 Player Alpha/Beta amplitudes in relaxed state(top), in anxious state (middle) , and in shocked state (bottom)

The stress emotion alters a material overlay for the players HUD (Heads Up Display), which results in a tunnel view and a blurred sight according to the level of stress the player currently experiences, as illustrated in figure 10.

The shock emotion triggers various effects in the game. One of them are the tendrils that become more aggressive, glow red and grow in size, thus hindering the players passage through them. The final opponent's attacks are also linked to this emotion, since they are only carried out when the player is shocked. There also exists a trap in the game, which the player has to trigger deliberately to get rid of an enemy standing on top of the trap.



Figure 10: Visual effect for anxiety parameter

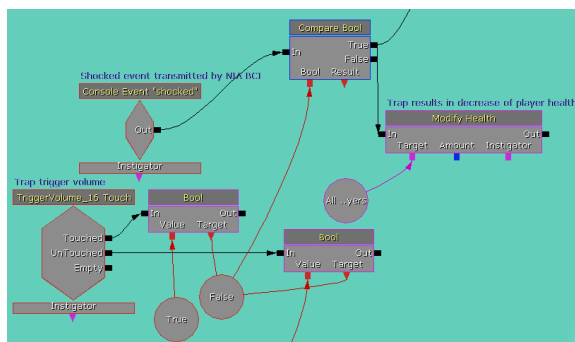


Figure 11: Kismet, UDK's graphical script editor

Unreal Development Kit (UDK)

The Unreal Development Kit (UDK) is a C++ based game development suite for Epic Games' Unreal Engine 3. Since the release of the first Unreal game it has continually been enhanced to meet the current technical state of the art. It offers the developer several tools and editor for almost every aspect of a game (see figure 11) while also implementing its own programming language named UnrealScript. The full version of the UDK is free to use for non commercial projects.

With the scripting framework and the graphical script editor Kismet it was possible for us to create own entities in the game that react to the inputs given from the NIA BCI.

5. RESULTS

Evaluation of the game

To evaluate our concepts and their implementations in the game we created a test scenario and let eight participants play through the entire game. We observed them while they were playing and asked them questions after they finished the game.

To support the immersion the participants had to play in a dark room with a 27 inch monitor in front of them. To have an adequate presentation of bass and other sounds we used a sound system with a sub

woofer. The sensors of the NIA headband on the foreheads of the participants had direct contact with the skin. In front of them they had a keyboard and a mouse. To ground the NIA, they had to lay their arm

on the NIA device. Before the game was started we adjusted the amplitudes of the brainfingers in the NIA Software to a balanced level so they were on a low level when the participants were relaxed and reached maximum values when the participants did strong movements or were shocked. To test this we scared them all of a sudden from behind.

All the participants stated that they like to play computer games and that they are familiar with games played in first-person perspective. Five participants had heard of BCIs before and two of them had tried them but none of them believed that it's possible to make controlled inputs via BCI.

All of the participants got scared during the game through audio visual effects (only this combination works significantly). This happened in a situations were a skull appears (figure 12) with a loud sound and when spikes come out of the ground and where a tree that works as a bridge over a canyon falls down. All of these situations are happening all of a sudden and address more than one stimulus modality of the player.



Figure 12: Shock sequence

All of the participants got immersed in the game. That was observable during shock moments and exclamations like "Ouch!" when they were attacked, "This is beautiful." during exploration, "I think I need a counterbalance here." while solving a puzzle and "Ha!" after an enemy was defeated. The participants stated that they were immersed the most when they were challenged to get through tendrils, to solve puzzles, to get used to a new situation and when the world reacted to their actions.

There also were situations which decreased the immersion. That is when something in the game world seems implausible. For example, the strange appearance of the enemy creatures and the partial exaggerated tunnel vision effect that reminded one participant more of icicles than a visual

representation of mental stress. Other situations were when the participants didn't know what to do next, when they had to think about a puzzle for too long and when they died.

The participants described the atmosphere as dark, eerie, mystic, mysterious, disturbing, thrilling, surreal and dense. As reasons for that they identified the dark presentation, field of view and blur effects, music and sound. They also mentioned the interplay between places and moods and the tendrils that reacted to the BCI.

The participants liked the fact that there are no weapons in the game and that artifacts like altars and gates stir their imagination. Their motivation to play through the game was their curiosity and the will to find a way back to civilization. Although the game never explains what it is all about, most of the participants drew the conclusion that they were knocked out after the crash with the hang glider and that it was all a dream.

Asked about the effects of the BCI, the participants answers were skeptical. Five participants stated that it worked partial while three participants couldn't tell that the BCI had any effect on the game. Maybe the reasons for that are that the NIA Game Controller doesn't work precisely and we weren't able to calibrate it exactly to fit with every participant. The NIA can't recognize which emotion or action caused a change of alpha or beta rhythms so the game reacts not only to mental stress or a shock moment but also to laughter, a cough and other things. Furthermore the players aren't able to associate every action that the BCI triggers with a brain activity that they are unconscious of. For example, one participant thought that the tunnel vision is a scripted event that always triggers with a shock sequence although it wouldn't be triggered at all without the BCI.

Concluding Remarks

Despite of the technical limitations of low-cost input devices, we were able to provide a proof of concept for emotion-based human computer interaction patterns and their use in a tailored game environment.

Emotional inputs may be evoked on demand, but it is by far more difficult to suppress them. Since the users in our experiments were aware of their emotional tracking, most of them recognized the impacts of their emotional input on the game progress, particularly regarding the opponents' actions. It should be noted that not being aware of these impacts does not necessarily indicate that these do not exist, as in real life also many cues are missed.

All in all, we conclude that emotional sensing has great potential for future game design, since it

provides the players' challenge of using and reflecting about their emotions in a systematic way.

Outlook

Since the appearance of cheap BCIs for the mass market, scientific studies and development costs for new applications and approaches for brain controlled systems have greatly dropped. This results in a raising interest of the industry, especially the game industry, since here new innovations are adapted very early to stand out from the huge count of competitors on this market.

Our evaluation concluded that all participants viewed the BCI as an improvement to the game experience and immersion of the players. This hints that further development should be considered, especially by enhancing the methods to correctly decipher player emotions.

One approach of enhancing these methods are the use of machine learning and neural systems. Murugappan et al. already conducted a promising study on these methods [Mur10a] that would make the process of correctly interpret user emotions much more precise.

Since the precision of control with a Brain-Computer-Interface as the only method is still very low, the use of a BCI as the only input modality would probably not cope (at least in the near future) with the complex control mechanisms of a modern computer game. Future BCI controlled games would probably implement a BCI as an additional modality.

Apart from the use as a game input method, emotion aware applications for Brain-Computer-Interfaces could theoretically fill one of the most important gaps in human-computer communication: The emotional level of communication. Emotions are a natural factor in human-to-human communication and make up a big part of the information transferred to the other. In communication between humans through a computer this problem is usually countered with the use of emoticons or smileys to give the other a help on how to interpret a message. A working emotion aware system could possibly solve this issue better and more precise. In a pure human to computer interaction the computer would have the means to understand the users intentions better and act accordingly.

6. ACKNOWLEDGMENTS

The authors would like to thank the University of Applied Science Bremen for the mentoring of this work, which is based on a bachelor thesis and the helpful community of the official epic forums for hints and tips regarding the UDK.

7. REFERENCES

- [Bro04a] Brown, E., Cairns, P. A Grounded Investigation of Game Immersion. CHI EA '04, New York, NY, ACM Press pp.1297-1300, 2004.
- [Car06a] Carpi, F., De Rossi, D., Non invasive brain-machine interfaces, European Space Agency, the Advanced Concepts Team, Ariadna Final Report (05-6402), pp. 4, 2006
- [Mat00a] Mateas, M. A neo-Aristotelian theory of interactive drama. Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, Palo Alto, CA, 2000.
- [Mur10a] Murugappan, M., Rizon, M., Nagarajan, R. et al. Inferring of Human Emotional States using Multichannel EEG, in: European Journal of Scientific Research, 48:2 (2010), S.281-29
- [Nic12a] Nicolas-Alonso, L., and Gomez-Gil. Brain Computer Interfaces, a Review, in Sensors, No 12, pp. 1211-1279, 2012
- [Nie99a] Niedermeyer, E., Da Silva, L.: Electroencephalography, Basic Principles, Clinical Applications And Related Fields, The Normal EEG of the Waking Adult, Baltimore, Lippincott Williams & Wilkins, 1999
- [Nij09a] Nijholt, A., Bos, D., Reuderink, B.: Turning shortcomings into challenges: Brain-computer interfaces for games, in: Entertainment Computing 1, pp. 85–94, 2009
- [Pfu99a] Pfurtscheller, G. and Lopes da Silva, F. Event-related EEG/MEG synchronization and desynchronization: basic principles, in Clinical Neurophysiology, No 110, pp. 1842–1857 1999
- [Rot09a] Roth, C., Vorderer, P., Klimmt, C. The Motivational Appeal of Interactive Storytelling in I. Iurgel u.a. (ed.) Second Joint International Conference on Interactive Digital Storytelling, Berlin et al., Springer pp. 38-42, 2009.
- [Tan10a] Tan, S. D., Nijholt, A. (Eds.). Brain-Computer Interfaces. Applying our Minds to Human-Computer Interaction, London, Springer, pp. 96, 2010
- [Wol02a] Wolpaw, J., Birbaumer, N., McFarland, D., et al. Brain-computer interfaces for communication and control, in: Clinical Neurophysiology, No. 113, pp. 767–791, 2002

Comics reading: An automatic script generation

Raulet Jérémy

LIASD
2, rue de la Liberté
93200, Saint-Denis,
France

jraulet@ai.univ-paris8.fr

Boyer Vincent

LIASD
2, rue de la Liberté
93200, Saint-Denis,
France

boyer@ai.univ-paris8.fr

ABSTRACT

With the advent of portable devices, reading comic ebooks is a popular activity. However, a simple scan of a comic page is not well adapted for portable device screens and a panel to panel reading without animations and adapted transitions is quite uncomfortable and not suitable. Moreover, applying manually transitions between each panel to script a complete comic book is a tricky task and seems impossible for a complete collection of comics. We present a model able to automatically script comics reading by using panel lines of force. Our results demonstrate that this model proposes a coherent solution for 87.2% of panels in an interactive time.

Keywords

Comics Script Generation, Comp.Vision & Image Processing, Mobile & WEB Graphics

1 INTRODUCTION

Nowadays, the number of comics novelty per year is in constant increase and reading them on a portable device is a common activity. These comic ebooks can be very different kinds, from a simple scan of a comic book to an electronic comic completely dedicated to the device screen and even a cartoon-like video.

Even if a comic especially created for a specific portable device seems to be the best solution, there is no appropriate solution to distribute them in an ebook format: other existing comics are scripted by a scriptwriter to produce input and output animations for each panel and exported to different portable devices. This work is performed in very different ways: by creating panel by panel transitions and animations using a dedicated tool [Rau11]; by creating a path in a comic page and displaying the entire page on the screen [Wan11]; or in the worst case, by creating a video of the comic.

We think that the first solution (i.e. creating transitions and animations panel by panel) is the best one to improve the reading experience without altering the content. However this solution is the most expensive and one can imagine how tricky the task is if the purpose is to process a comics library. Thus its automation is an

interesting challenge both for researchers and commercial comics publishers.

In this paper, we consider panels reading and panels transitions. The panel extraction is realized as a preliminary step with for example Yamada et al. [Yam04], Tanaka et al. [Tan07] or Raulet et al. [Rau11] methods. For each panel, we aim at proposing an input and output animation based on its reading direction.

First, we present the terminology and the specificities of comics which are used to identify possible solutions. Then we present the related work on image retrieval and interest point detection considering the specific topics (i.e. panels transitions and reading). Then we propose our model based on image processing techniques. Results are provided comparing related work and our model. Finally, we conclude and propose future work.

2 TERMINOLOGY

In this section, we present the terminology used throughout this paper. Hereafter we precise the context and give our definitions but we do not attempt to provide an exhaustive study on comics. The interested reader should refer to [McC93] and [McC00] and as there is not a unique and unambiguous definition for all of these terms, one can find a part of this vocabulary on the website [Comi09]. From global to detail and according to our definitions, we also describe the noteworthy variations in comics to illustrate the wide range of possibilities.

Usually, a comic is described by a succession of **pages** composed by a set of image **strips**. These images, named **panels**, are colored or black and white and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

are often separated by **gutters**. Remark that since Rodolphe Töpffer in 1830, considered to be the modern comics creator, this page composition has been constrained to the artist by the publishing world.

Like Scott McCloud [McC93], we consider that a comic is a succession of panels. Each of them have their own size and form and are often surrounded by a black **border**. **Open panel** depicts panel without any borders. In case of overlapping between two or more panels the **overlapped panel** term is used. A panel frequently contains **speech balloons** and/or **captions** describing respectively the dialogue and the scene.

Even if this terminology covers american comics, manga, franco-belgian comics, graphic novels and all other styles, it is not enough to create a taxonomy of the domain. Many differences exist between these styles (see figure 1) depending on many factors: the technique used (brush, pencil...), the authors (two comics of the same author can be radically different)... Even for a given comic the visual representation of characters, scenes, places, that must be unique, may vary. Due to these variations, admissible for any comic readers, and the number of characters, it is not possible to build a comic database representing the collection of characters and uses it to describe the movement.

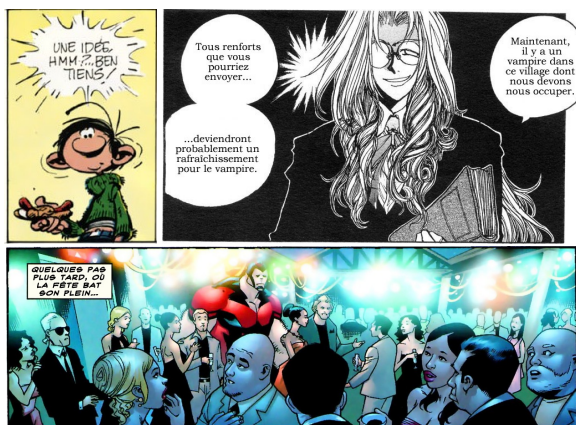


Figure 1: Top left: panel of *Gaston Lagaffe*, Top right: panel of *Hellsing*, Bottom: panel of *X-men*. These panels represent respectively franco-belgian comics, manga and american comics with different styles, levels of details and colors.

If a comics classification is not possible, one can focus on the different transitions between two successive panels and try to determine their graphics impacts.

Scott McCloud, in [McC93], has defined six forms of transition:

1. **moment-to-moment**: The second panel represents the scene a little time after the previous one, like if two photographs have been taken with a second of interval;

2. **action-to-action**: The next panel represents the next action, like a selection of key moments describing a story (see figure 2);
3. **subject-to-subject**: The same idea is illustrated in the two panels but no direct visual relation exists like in action-to-action. A common example is a phone discussion between two characters in which each panel represents a character in its own environment;
4. **scene-to-scene**: Time or distance is clearly visible between the two panels. A landscape in summer and the same in winter is an example of scene-to-scene transition;
5. **aspect-to-aspect**: The two panels describe the different aspect of the same idea or place at the same time: a beach and a character in swimsuit;
6. **non-sequitur**: No logical relation exist: suppose that figure 1 is a comic page composed by these three panels.

All of these transitions may be found in the same comic, even if the sixth is uncommon. The moment-to-moment transition is the one where panels are the most similar. But even in this case, the artist may redraw the entire panel and change, voluntary or not, a large part of it (see figure 2). It is possible that a reader does not take care about those differences but they exist. One of the most visible is the position and size of speech balloons which obscure the background.



Figure 2: Page 20 of *Asterix and the Secret Weapon* panel 2 and 3 of the first strip. Excepted characters, there are many changes between these two panels. The main change is the house behind Asterix in the left panel, that disappears in the second one.

In a page, the **reading direction** is left top to right bottom, excepted in a manga. It influences the reading direction of a single panel and the eye movement should begin at the top left corner and follow a **Z pattern** in most cases.

This expected movement is disturbed by all panel elements. For example, as explained by Omori et al. [Omo04], a reader frequently skips a panel without any speech balloon. In the component hierarchical theorist, Almasy [Alm75] has explained the importance of living subjects for the reading direction. This means that the reader does not just follow a Z pattern but search important elements into the panel, like a

character. Artists have several other ways to direct the reader's eyes: color contrasts, object size, level of details, closeup... In fact, comic creators determine the reading direction while generating each panel.

With the widespread of comics during the last century, artists have become accustomed to use these techniques to give the wish of pull up the reader's eyes on the second page of a double page in a comic book. They also use them to encourage the reader in turning the page after the last panel of the double page. But nowadays, panels are not necessarily arranged in a page. For example, on a mobile phone, the reader can watch each panel one by one and can have eye movement into a panel but not between two of them. Thus we need to find new techniques to direct the reader's eyes and let him concentrate on the story and not on transitions between panels.

3 RELATED WORK

To produce animation for panel transition it is necessary to detect similar contents and transformations between two consecutive panels. We have identified two main approaches:

1. Image retrieval, to detect and follow objects in a panel sequence;
2. Interest point detection and comparisons of their position to interpolate movements between panels.

Hereafter, we focus related work on these two approaches giving their advantages and drawbacks.

3.1 Content-Based Image Retrieval (CBIR)

In [Tor06], Torres et al. have explained the CBIR theory which, in particular, allows to index images with a distance function and to distinguish objects with their shape descriptors. In all CBIR methods, the main idea consists in the similarity and difference evaluation between two images.

Landré et al. [Lan07] have proposed a CBIR method using a Hamming distance and a query-by-visual-example method to compare shapes. In order to have a better perception of distances between colors, images are represented in the Lab colorspace. Then, three binary signatures per image for color, texture and shape (with a laplacian edge detector) are computed. Finally, similarities between images for each signature with a Hamming distance (a XOR binary operator) are searched. This method is well adapted to find images with the same theme (a red flower for example) and works well in general, but it is imprecise and cannot, for example, distinguish two human characters. Remark that it is possible that this method works well

for a moment-to-moment transition or maybe action-to-action but it is impossible for subject-to-subject transition. Moreover, this method uses colors and some comics are "just" black and white.

The approach proposed by Fekir et al. [Fek09] is based on a Region Of Interest (ROI). This ROI is selected with a circle snake on the first image of the sequence. Then, on each image of the sequence, energies (curve consistency, gradient...) are minimized and the snake is moved. Finally, this new snake is treated like an automatic initialization on the next image and the second step (i.e. energy minimizations and snake movement) is repeated. This approach is used to follow cells in a sequence of echocardiographic images. Unfortunately, except for the moment-to-moment transition, differences between two panels are too important to implement this kind of method.

Cheung [Che07] has developed an application named MAIRE to recognize a human-like character face that helps the reader to find a particular scene in a large collection of comics. First, he has proposed the use of two CBIR methods for face detection and recommended the Adaboost one. Then, he has implemented four face recognition methods and proposed to use the EBG (Elastic Bunch Graph Matching). These two steps enable to sort panels depending on present characters and allow the user to perform a query to find a particular scene into a large database of comics. Unfortunately, this approach requires a database of characters and as explained in section 2, it is impossible to be exhaustive. Moreover, even if it is not carefully mentioned in the paper, the detection seems to work only on full-frontal faces.

3.2 Interest Point Detection

In [Sch00], Schmid et al. have introduced two criteria for the evaluation of interest point detectors: first, the repeatability, allowing to compare the position of interest points in two images of a scene; second, the information content, allowing to measure if an interest point is really distinct one from another. They have concluded that Harris detector is the best solution for these two criteria. This method seems suitable to our problem of detecting interest points in a panel and like SIFT, SURF and ORB are posterior to [Sch00]. We present hereafter these four methods.

Gabriel et al. [Gab05] have proposed a method based on an improved implementation of Harris detector to follow an object in an image sequence. First, for each object to be tracked, a ROI is defined. Then, each ROI is described by interest points obtained from the colored version of Harris detector. Finally, the object is found in the next image with a comparison of the relative positions of interest points. The problem is that this method works with images without significant change

and for any forms of transitions excepted a moment-to-moment transition we cannot initialize the ROI on each panel.

Bauer et al. [Bau07], have compared SIFT and SURF detectors. They have evaluated the invariance against rotation, scale, noise, change in lighting condition and change of view point on images of natural outdoor scenes. They have concluded that SIFT has the best performance in term of repeatability but followed very closely by SURF. They have also concluded that SURF produces fewer points and the comparison is faster. This comparison is done on photorealistic images only. We think that these methods have a bad repeatability in our case due to the precision of drawings and the difference between two similar panels. Even if our model is not based on this kind of method, we have implemented it and present benchmarks in section 5.1 to confirm our hypothesis.

Rublee et al. [Rub11], have recently presented an efficient alternative to SIFT or SURF named ORB (Oriented FAST and Rotated BRIEF). FAST is used to detect key-points and BRIEF to describe it. It seems more efficient and faster than SIFT and SURF but like [Bau07], only photorealistic images have been tested to provide benchmarks. Like SIFT and SURF, ORB is shown efficient for their experiments but has not been tested on expressive images. Our model is not based on this method but we have implemented it and present benchmarks in section 5.1.

We have presented several approaches to extrapolate a movement between two panels and no one is adequate for all transition forms. The two main problems of these approaches are:

- Methods are dedicated to follow objects in a sequence with little modification between two images;
- Methods have been evaluated only on photorealistic images.

We propose our model that enables to extrapolate a reading direction for a given comic panel.

4 MODEL

We present our model dedicated to decide both panel reading direction and panel transition. As detailed in previous work, approaches that may provide panel transitions do not exist and photorealistic approaches cannot be adapted to this kind of problem. Thus rather than a top-bottom approach providing first panel transitions to deduce the reading direction, we prefer a bottom-top approach providing first panel reading direction to deduce panels transitions. Since a comic panel is the result of an artistic process, our solution consists of determining artistic elements providing a reading direction for each panel. For that reason, our approach is based

on the image processing techniques being able to collect information available in each panel. Our process is realized in 3 main steps:

1. (a) To provide a solution for any panels of any comics (i.e. colored and/or black and white), we perform an edge detection on the panel and use this information only (i.e. no color information are used hereafter);
 (b) Based on this edge detection, we extract lines of force providing a large set of possible reading information;
2. We improve our lines of force research by focusing only on dynamically defined ROI panel by panel. Thus, we keep only the most interesting part of them;
3. A classification system is finally used to determine the panel reading direction. Possible reading directions are horizontal (from left to right), vertical (top to bottom) and the two diagonals (from left to right).

Finally, according to reading directions of two consecutive panels and rules given by the scriptwriter, we provide automatically panel transitions. In practice, rules are associations between the directions and the panel transitions. These are realized independently by the scriptwriter and can be reused or changed for any comics.

4.1 Edges and Lines of Force

As a first step, we extract edges and lines of force in each panel. Lines of force is a graphical technic used since the renaissance period and are intended to convey the directional tendencies of object through space. We combine two image processing techniques to provide lines of force: an edge detection and a feature extraction technique.

As the most common edge detectors (Sobel, Prewitt, Canny) are almost interactive, we prefer the Canny detector for its detection performance [Sha02]. A Sobel kernel filter is used in the Canny detector and experiments show that a 3×3 kernel filter is the most appropriate kernel size. Other kernel sizes (i.e. 5×5 and 7×7) give a too detailed result. We follow the Canny's recommendation for the upper and lower thresholds and apply a ratio of 2:1.

Then, we use the Hough transform, as a feature extraction technique, to search the longest straight lines. We search a limited number of lines to avoid false positive with only a few lines and unfeasible results containing too many lines. This interval has been determined by a simulated annealing algorithm [Kir83] and must be in [30, 50]. These lines represent image lines of force which suggest the scene orientation. Depending on the comic style and the scene, straight lines may have different lengths.

In our algorithm (see algorithm 1), initial Canny thresholds values are used and dynamically modified according to the Hough transformation result; the Hough transform threshold is dynamically changed until the result converges to the attempted values in term of number of lines as follow: while we have not enough lines we decrease the minimal size of a straight line (Hough threshold). If the Hough threshold is too small, we decrease the Canny thresholds and repeat the Hough transform. While we have too many lines, we increase slowly the Canny thresholds. This produces a set of lines representing the panel lines of force (see figure 3).

Data: panel

Func: image Canny (*imageSrc*, *lowerThreshold*, *upperThreshold*, *SobelFilterSize*);

Func: setOfLines Hough (*imageSrc*, *threshold*);

Result: LINES (lines of force set)

thresholdCanny \leftarrow 401;

minNbLine \leftarrow 30;

maxNbLine \leftarrow 50;

maxThresholdHough $\leftarrow \frac{3}{4}$ panelDiagonal;

minThresholdHough $\leftarrow \frac{\text{MIN}(\text{panelWidth}, \text{panelHeight})}{10}$;

repeat

dst \leftarrow Canny (*panel*, *thresholdCanny*, *thresholdCanny* \times 2, 3);

thresholdHough \leftarrow maxThresholdHough;

repeat

LINES \leftarrow Hough (*dst*, *thresholdHough*);

thresholdHough \leftarrow thresholdHough-1;

if thresholdHough \leq minThresholdHough **then**

thresholdCanny \leftarrow thresholdCanny-100;
break;

end

until nbLine < minNbLine;

if thresholdCanny \leq 0 **then**

break;

end

thresholdCanny \leftarrow thresholdCanny+5;

until nbLine > maxNbLine **OR** nbLine < minNbLine;

Algorithm 1: Lines of force detection.

However, the detected lines of force are, in most cases, disturbed by the border and speech balloons, so we propose a method to improve this result.

4.2 ROI

In figure 3, one can note that panel borders and speech balloons also produce lines of force. Since both are generally composed by straight lines, their impact on the line of force detection is very important. To avoid the noise generated by borders, the region on which our

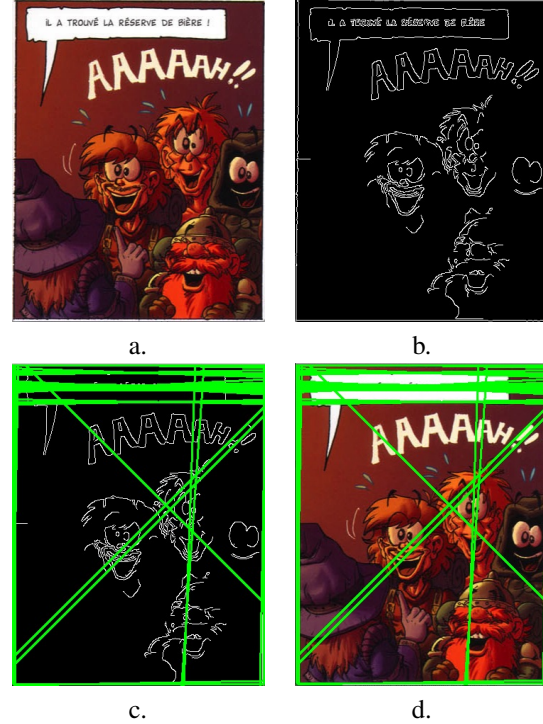


Figure 3: a) A panel of *Le donjon de Naheulbeuk*. b) Canny edges detector on (a). c) Lines of force with Hough transformation on (b). d) Latter lines on (a).

algorithm is applied is reduced by 10% on left, right and bottom of the panel. This value guarantees that borders will be removed and does not affect line of force detection as shown in our experimentation.

Comic artists follow some rules when creating speech balloons: they are often close to speaking character faces which are frequently located in the center of the panel; they are located where they do not hide a significant part of the drawing: for example it is uncommon that a speech balloon mask a part of a character's face; comic artists use the rule of third to place important elements in the panel and speech balloons are commonly located at the periphery. According to these principles, speech balloons are frequently placed on the top of the panel, in the sky or the scenery. In the case where speech balloons are in a particular layer, we do not consider them for the line of force detection. In any other cases we remove the upper part of the image according to the rule of third.

Figure 4 presents the line of force detection on a reduced ROI. As one can see, borders and speech balloons are not considered any more and the result is more relevant.

Now, these lines can be classified to extrapolate the reading direction.

4.3 Classification and Interpretation

We propose a classification and an interpretation system for the lines of force previously detected. We consider

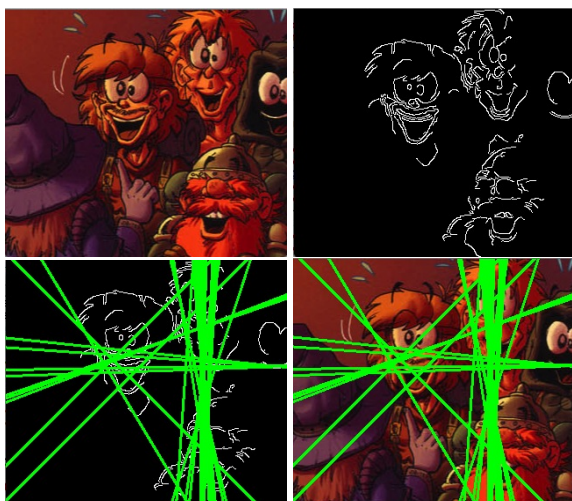


Figure 4: Figure 3 with a reduced ROI.

that the reading direction depends on the most representative direction of the lines of force.

First, we calculate the non-oriented gradient for each line. Then, lines of force are classified into 5 groups depending on their gradients: horizontal, vertical, the two diagonals and others. We fix a precision of $\pm \pi/16$ radian compared to horizontal and vertical axis and an angle of $\pm \pi/4$ radian for the diagonals. Others lines which are not classified in one of these four groups constitute the group named “other”. Figure 5 presents these groups. Remark that the surfaces of the four groups are equivalent and sum to the surface of the group named other (i.e. half of the circle surface). The figure 6 illustrates this classification on the example used throughout the article.

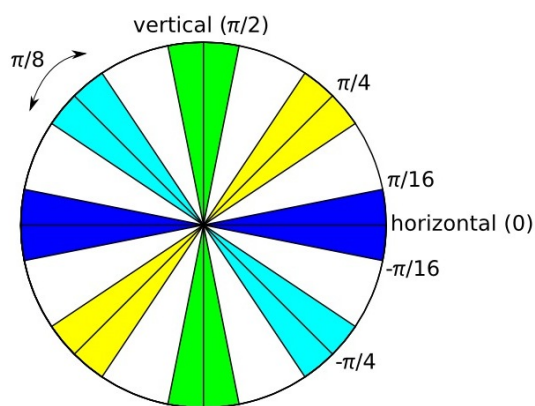


Figure 5: Lines of force classification. Each group is represented by a color and has an angular distance equals to $\pi/8$. The group named other is in white.

To select the panel reading direction, we sort all of these groups (except group other) according to the number of lines they contain. Finally, we compare the larger group to the total number of lines in two steps: first, if the group contains more than 33% of the lines, this group

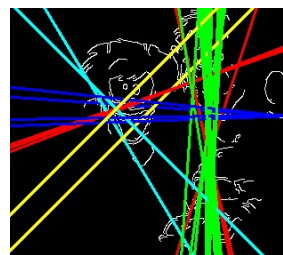


Figure 6: Each line color represents a group, the group other is in red.

is chosen to become the reading direction; otherwise, if this group contains more than 25% it is retained. Note that if the second larger group contains also more lines than the percentage that permits the choice of reading direction, it is chosen as a direction applicable if the panel is too large to be displayed in full screen (see algorithm 2 for more details).

Data: *setoflines*

Result: readingDirection optionalDirection

$$N \leftarrow |\textit{setoflines}|;$$
foreach *Line* in *setoflines* **do**

```

find gradient of line;
classify Line according to its gradient;
/* 5 groups:  horizontal,
vertical, two diagonals, other */

```

end

threshold $\leftarrow N/3$;

```
/* In all following conditions, we
do not test the group other      */
sorting groups in decreasing order;
```

if $|first\ group| > threshold$ **then**readingDirection \leftarrow group orientation;

else

threshold \leftarrow N/4;**if** $|first\ group| > threshold$ **then**| readingDirection \leftarrow group orientation;

else

```

| return unclassified;

```

end

end

if $|second\ group| > threshold$ **then**

optionalDirection \leftarrow group orientation;

```

return readingDirection and optionalDirection;

```

else

```

return readingDirection;

```

end

Algorithm 2: Lines of force interpretation.

Even if our system provides a classification for the most of the panels, some of them remain unclassified (see section 5.2). These cases occur when lines of force are mainly classified in the group other or when groups are balanced. Since we decide to provide one of the attempt

reading directions (horizontal, vertical and the two diagonals), for these cases, the reading direction is finally given by a scriptwriter.

Note that rules can be added or changed in our system implementation to interpret the reading direction. As an example, one might want to add a flicker animation if the two diagonal groups are the majority.

5 RESULTS

In this section, as mentioned in section 3.2, we focus first on results using SIFT, SURF and ORB. Then we present results provided by our model.

5.1 SIFT, SURF and ORB

As described above, these methods are well adapted for photorealistic image retrieval and have their own advantages and drawbacks. One can easily imagine to adapt these techniques to the comic panel transitions by finding corresponding points in successive panels. As we have evaluated carefully these methods on comic panels and have remarked that they are not dedicated to this kind of images, we provide hereafter salient results on panel transitions. However, for clarity purposes, we only focus on a single couple of representative panels where the transition is the most favorable (i.e. moment-to-moment transition).

Note that, as an implementation detail, we use OpenCV library for SIFT, SURF and ORB and we test the feature matching with flann, fern and brute force.

Figure 7 presents results of interest points matching on a panel. As one can remark, speech balloons and onomatopoeia produce noise during the interest points detection. Figure 8 illustrates the same algorithms using the ROI defined in section 4.2.

As concluded by Bauer et al. [Bau07], SIFT produce too many points to be easily readable. On top of figure 8 there are numerous good detections and matchings but also numerous false positives.

By opposition, SURF gives a very poor repeatability of detection and it is impossible to provide a confident panel transition (see middle of figure 8).

As mentioned by Rublee et al. [Rub11], ORB is a good compromise between SIFT and SURF. ORB (see bottom of figure 8) has a better repeatability than SURF but also fewer points compared to SIFT. However, here also, matching is still shoddy like for SURF and SIFT.

As mentioned in section 2 difference between two successive panels are often more important than between two photographs of the same scene (like in [Bau07] and [Rub11]). Also, since a large part to the repeatability is distorted by many changes between two panels (i.e. shapes, colors...), this kind of methods cannot be applied to comic images.



Figure 7: Up to down: SIFT, SURF and ORB detectors on two panels of comic *Le Donjon de Naheulbeuk*.

5.2 Results of Our Model

We experiment our model on a large collection of different type of comics: comic books (*X-men* and *Star Wars*), Franco-Belgian comics (*Gaston Lagaffe*, *Asterix*, *Tintin* and *Le Donjon de Naheulbeuk*) and mangas (*Naruto*, *One Piece* and *Hellsing*). They constitute a set of 2,000 panels. Our system has classified the panel set in 16 minutes with a Intel Core 2 Duo processor (2.26GHz) and 2Go RAM.

87.2% of the panels have been classified into one of horizontal, vertical or the two diagonals groups. 12.8% of panels are classified in the group other. This classification is presented in table 1; the horizontal reading direction is a majority with a distribution of 54.5%. This can be explained by the horizontal reading direction and the page format where panels are more frequently horizontal than vertical. The vertical reading direction is not well represented, which can be explained by the fact that few panels are vertically extended. Remark that our system has proposed the same reading direction (vertical) for the two panels used on figure 9.

optional direction		Horizontal	Vertical	Diagonal $\pi/4$	Diagonal $-\pi/4$
reading direction					
Horizontal	54.5%	N/A	0.9%	0.9%	1.4%
Vertical	8.5%	0.6%	N/A	0.8%	0.7%
Diagonal $\pi/4$	14.2%	1%	0.7%	N/A	0.6%
Diagonal $-\pi/4$	10%	0.5%	0.3	0.6%	N/A
Other	12.8%	N/A	N/A	N/A	N/A

Table 1: Summary of the interpretation of reading direction on 2,000 panels. Column 2 is a total for a given direction. Columns 3 to 6 correspond to the optional direction, applicable if the panel is too large to be display in full screen.

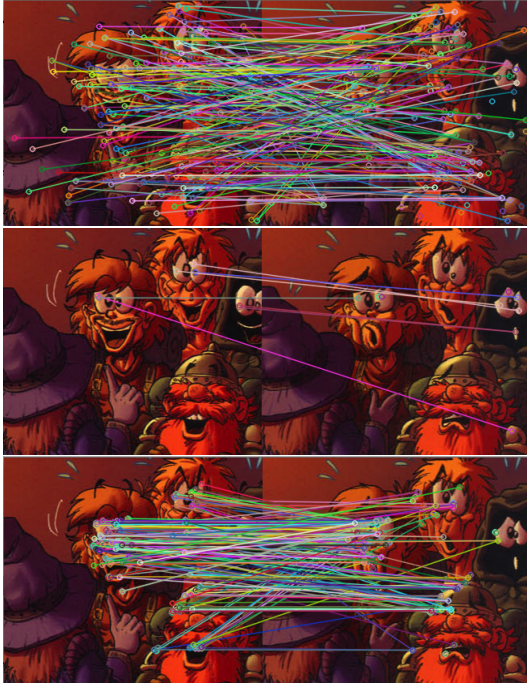


Figure 8: Up to down: SIFT, SURF and ORB detectors on ROI of two panels of comic *Le Donjon de Naheulbeuk*.



Figure 9: Lines of force of two successive panels of comic *Le Donjon de Naheulbeuk*. On these two panels, our system has proposed a vertical reading direction.

Note that only 10% of the panels (excepted panels in the group other) have an optional direction. That means that the difference between the first and the second orientation of a panel is generally large enough.

As our test protocol consists of a comparison between results provided by our model and by a panel of hu-

mans testers, we have selected randomly 100 panels in our set. Note that testers are comic readers (27%), game designers (13%), graphic designers (20%) and other (40%). We asked them to select only one group for each comic panel and we retain only the majority group. As one can see in table 2, reading directions proposed by testers and our system are very similar. The first column represents results with our tool for these 100 panels. Our results and human choice are concordant in 78% of the cases (91% for horizontal, 100% for vertical, 56% for $\pi/4$ diagonal, 46% for $-\pi/4$ diagonal and 41% for others). Remark that for horizontal and vertical, the results are identical in more than 93% of the cases. Note that, even if it does not appear in this table, the answers provided by testers for each comic panel, are generally distributed uniformly with a dominant group.

6 CONCLUSION

We have proposed a model allowing to script a comic ebook reading by adding panel reading direction and inputs/outputs animations. Our method has classified 87.2% of panels in the same way a human would do in 78% of cases. The purpose of this work does not consist of replacing scriptwriter but to suggest animations to them and to reduce the time of a comic script generation. Our comics reading system is integrated to a very complete sketch-based interface to script comics reading. In future work, we plan to use curved lines as lines of force and we aim at integrated perspective to propose more complex animations.

7 ACKNOWLEDGEMENT

This work was supported by TEKNEO (producer of entertaining applications (video games), as well as serious applications (Serious games)). J  r  my Raulet was a recipient of a CIFRE fellowship.

8 REFERENCES

- [Alm75] P. Alamy. *La Photographie, moyen d'information*. Tema  ditions, 1975.

human choice		Horizontal	Vertical	Diagonal $\pi/4$	Diagonal $-\pi/4$	other
our tool						
Horizontal	48	44	3	1	0	0
Vertical	12	0	12	0	0	0
Diagonal $\pi/4$	16	2.33	4.33	9	0	0.33
Diagonal $-\pi/4$	12	0	3	0	8	1
Other	12	2	1.5	0.5	3	5
total of human choice		48.33	23.83	10.5	11	6.33

Table 2: Summary of the interpretation of reading direction on 100 panels. Column 2 is a total for a given direction for our tool. Columns 3 to 7 show testers majority direction choices compared to results with our tool.

- [Bau07] J. Bauer, N. Sünderhauf, and P. Protzel. “Comparing several implementations of two recently published feature detectors”. *Proc. of the International Conference on Intelligent and Autonomous Systems*, Vol. 6, No. pt 1, pp. 143–148, 2007.
- [Che07] S. C. S. Cheung. “Face Detection and Face Recognition of Human-like Characters in Comics”. Tech. Rep., City University of Hong Kong, Department of Computer Science, 2007.
- [Comi09] “Comicbook Dictionary”. Oct 2009. <http://comicbooks.wikidot.com/comicbook-dictionary>.
- [Fek09] A. Fekir, N. Benamrane, and A. Taleb-Ahmed. “Détection et suivi d’objets dans une séquence d’images par contours actifs”. *Proc. CIIA*, Vol. 547, 2009.
- [Gab05] P. Gabriel, J.-B. Hayet, J. Piater, and J. Verly. “Object tracking using color interest points”. In: *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pp. 159–164, IEEE, 2005.
- [Kir83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Optimization by simulated annealing*. Vol. 220, Washington, 1983.
- [Lan07] J. Landré and F. Truchetet. *Image retrieval with binary hamming distance*. Citeseer, 2007.
- [McC00] S. McCloud. *Reinventing Comics: How Imagination and Technology Are Revolutionizing an Art Form*. HarperCollins, 2000.
- [McC93] S. McCloud. *Understanding Comics: The Invisible Art*. Tundra Publishing, 1993.
- [Omo04] T. Otori, T. Igaki, T. Ishii, K. Kurata, and N. Masuda. “Eye catchers in comics: Controlling eye movements in reading pictorial and textual media.”. *28th International Congress of Psychology*, pp. 211–219, 2004.
- [Rau11] J. Raulet and V. Boyer. “A Sketch-based Interface to Script Comics Reading”. *SIG-GRAPH Asia 2011 Sketches*, p. 3, Dec 2011.
- [Rub11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. “ORB: an efficient alternative to SIFT or SURF”. *International Conference on Computer Vision*, pp. 2564–2571, Nov 2011.
- [Sch00] C. Schmid, R. Mohr, and C. Bauckhage. *Evaluation of Interest Point Detectors*. Vol. 37, Springer, 2000.
- [Sha02] M. Sharifi, M. Fathy, and M. T. Mahmoudi. “A Classified and Comparative Study of Edge Detection Algorithms”. *Information Technology: Coding and Computing*, pp. 117–120, 2002.
- [Tan07] T. Tanaka, K. Shoji, F. Toyama, and J. Miyamichi. “Layout Analysis of Tree-Structured Scene Frames in Comic Images”. In: *Proceedings of the 20th international joint conference on Artificial intelligence*, pp. 2885–2890, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
- [Tor06] R. d. S. Torres and A. X. Falcão. “Journal of Theoretical and Applied Informatics (RITA)”. *RITA*, Vol. XIII, No. 2, pp. 165–189, 2006.
- [Wan11] A. D. Wandani, G. Wee, and W. S. Moses. “Designing Interactive Mobile Comics for Multi-Touch Screen Phones”. *International Conference on Future Information Technology IPCSIT*, Vol. 13, pp. 332–336, 2011.
- [Yam04] M. Yamada, R. Budiarto, M. Endo, and S. Miyazaki. “Comic Image Decomposition for Reading Comics on Cellular Phones”. *IEICE Transactions*, Vol. 87-D, No. 6, pp. 1370–1376, 2004.

Practical Augmented Visualization on Handheld Devices for Cultural Heritage

Giovanni Murru
Sapienza Univ. of Rome
giovanni.murru@gmail.com

Marco Fratarcangeli
Sapienza Univ. of Rome
frat@dis.uniroma1.it

Tommaso Empler
Sapienza Univ. of Rome
tommaso.empler@uniroma1.it

ABSTRACT

In this paper, we present a framework for the interactive 3D visualization of archaeological sites on handheld devices using fast augmented reality techniques. The user interface allows for the ubiquitous, personalized and context-aware browsing of complex digital contents, such like 3D models and videos. The framework is very general and entirely devised and built by the means of free, cross-platform components. We demonstrate the flexibility of our system in a real case scenario, namely the augmented visualization of a historically reliable model of the Ancient Forum of Nerva located in Rome, Italy.

Keywords

Virtual and augmented reality, personalized heritage visits, mobile guides, location-aware.

1 INTRODUCTION

Augmented reality (AR) is an emerging computer technology where the perception of the user is enhanced by the seamless blending between real environment and computer-generated virtual objects coexisting in the same space. The resulting mixed image supplements reality, rather than replacing it [7].

In the context of cultural heritage, augmented reality is used to blend visual representations of historical monuments, artifacts, buildings, etc., into the real environment visited by the audience (e.g., tourists, students, researchers). For example, in virtual Pompeii [15], virtual characters representing ancient Romans are blended into the real environment; the user is able to perceive them by means of an ad-hoc mobile AR system. Such applications create simulations of ancient cultures by integrating them in the actual real environment. In this way, the user can learn about the culture by directly interacting with it on site.

Augmented reality systems are rather complex and involve technologies from different areas such as computer vision, computer graphics and human-computer interaction, in order to synthesize and deliver the virtual information onto the images of reality. Generally, the system must 1) track and locate the real images, 2) display the virtual information and 3) align and su-

perimpose the virtual data onto the real image. The main challenge in the design of these systems lies in the seamless integration of computationally-expensive software modules and energy consuming hardware in a framework that must run at interactive rate and, at the same time, be portable by the user. Traditionally, the mobile systems [19, 15] require the user to wear a set of hardware devices such as cameras, electronic compasses, small laptops, which makes the whole system not comfortable and limits *de-facto* the massive spreading of mobile AR systems in the context of cultural heritage.

The recent increase of the computational capabilities, the sensor equipment and the advancement of 3D accelerated graphics technologies for handheld devices, offer the potential to make the AR heritage systems more comfortable to carry and wear, facilitating the spread of this kind of AR systems to the mass market.

1.1 Contributions

In this paper, we present a novel mobile framework for augmented reality in the context of cultural heritage, running on modern handheld devices. The framework implements context-aware tracking, 3D alignment and visualization of graphical models at interactive rate. Using this framework, the user is free to roam around archaeological sites using non-invasive and already in use devices such as modern smartphones and tablets. The framework is composed by free, cross-platform software modules, making it easier to reproduce.

The applicability of the framework is tested by providing an augmented view of the Ancient Forum of Nerva, which was one of the Imperial Fora during the Roman Empire age. The 3D model has been designed accord-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

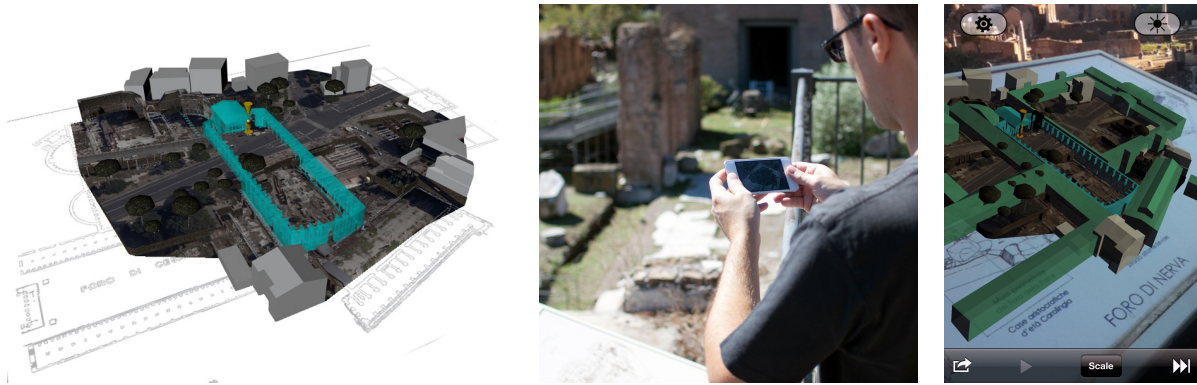


Figure 1: *Left.* 3D reconstruction of the Ancient Forum of Nerva in Rome, Italy. *Middle.* Augmented view of the archeological artifact on-site. *Right.* A screenshot of the framework running on a handheld device.

ing to the information acquired from previous archaeological studies [18].

2 RELATED WORK

Augmented Reality is a technology allowing for extending the vision of real world with superimposition of digital information, e.g. 3D virtual objects, 2D images and icons, labels, etc. Augmented reality is not intended for replacing the reality like traditional virtual reality, but it rather enhances it with digital data, making virtual and real objects coexist in the same space. In general, an augmented reality system must be equipped with display, tracker, graphics capabilities and appropriate software [6, 7, 9].

Head-Mounted Displays [13] are one of the most popular approaches for delivering mobile augmented reality in the context of cultural heritage. The Archeoguide project is among the pioneer systems for the on-site exploration of outdoor sites [11, 19]. Such a system is able to track position and orientation of the user employing a camera and an electronic compass, both mounted with the display unit. This allows for inferring the field of view and displaying 2D virtual images and information of the scene observed by the user. However, the weight and dimension of the required hardware devices makes the whole system uncomfortable to wear.

Modern handheld devices, such like smartphones and tablets, have a complete set of high quality sensors such as 3-axis gyroscope, ambient light sensor, accelerometers, magnetometer, proximity sensor, and assisted GPS; hence they are well suited for the development of augmented reality systems in a scenario similar to Archeoguide. Recent research efforts have provided several mobile solutions [8, 12, 20, 10] but are still not able to visualize context-aware, outdoor 3D models in a general manner.

Existing commercial mobile applications in the context of cultural heritage touring [3, 4, 1] lack of the capability to interactively blend 3D content to the observed real

scene. In general, they completely replace the video feed coming from the camera with a 3D scene, requiring the user to stand in a known position and implementing a simple alignment based on the compass of the device, without an actual tracking and registration of the video feed with the virtual objects as it happens in augmented reality systems.

3 OVERALL DESIGN

In an augmented reality system, objects in real and virtual worlds must be precisely aligned with respect to each other, otherwise the illusion that the two worlds coexist is compromised. An augmented reality system lacking of accurate registration is not acceptable [6], and this is a requisite for establishing a connection between the features of 2D images of reality and the 3D virtual world frame [9]. Computer vision is extensively used in AR systems mostly for two main tasks: image tracking and reconstructing/recognizing. Tracking algorithms interpret camera images and, in this context, can be categorized in two classes: those relying only on feature detection and those using a precomputed model of the tracked object's features.

The framework employs the markerless tracking capabilities of the Vuforia SDK [5] to compute in real-time the position and orientation (pose) of some predefined image targets located in the real environment. Once the pose is retrieved, a virtual scene is rendered, overlaid and registered onto the video camera feed by using the OpenSceneGraph module [21].

Hence, the proposed framework is based on the communication between two main software components, namely Vuforia and OpenSceneGraph, providing a touch interface to the user (see Fig. 2). Vuforia is the module responsible for tracking an image target and estimating its pose in 3D space, while OpenSceneGraph is the one delegated to manage the rendering of the models. The most important features of these packages, and how they are employed within the framework, are briefly depicted in the following sections.

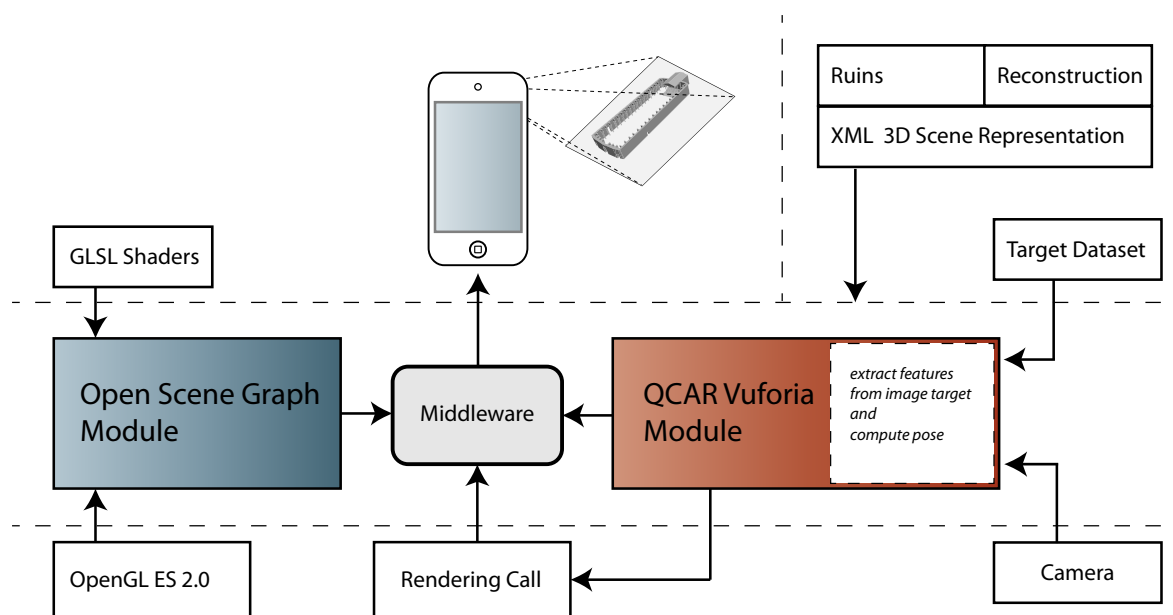


Figure 2: Schematic view of the framework components.

3.1 Vuforia SDK

Vuforia by Qualcomm [5] is a mature platform aimed to a wide range of handheld devices, supporting both iOS and Android; it is released for free, even for commercial purposes. Within our framework, Vuforia is used for the markerless tracking of the camera images. Feature based representations of relative user-defined *image targets* are created for each monument.

To create these targets, the developer should upload the image that needs to be tracked on Qualcomm servers by using a simple web interface, namely the Target Management System. Afterwards the developer can download the relative target resources bundled in a dataset, which can then be loaded at runtime through a simple API call. Such a dataset contains an XML configuration file allowing the developer to configure certain trackables' attributes, and a binary file containing a representation of the trackables.

The accuracy of the tracking is based on the number and the quality of the relevant features extracted by the Target Management System. For achieving a good tracking quality, input images must 1) be rich in detail, 2) have optimal contrast and 3) avoid repetitive patterns like grassy fields, the facade of modern house with identical windows, a checkerboard and other regular grids. Once built, the datasets are loaded during the application initialization. Only one dataset can be active at a certain moment, however a dataset can contain multiple targets.

Our framework handles the following core components of Vuforia:

Camera. The camera component manages the capture of each frame from the device camera, and transmits the

data to the tracker. The camera frame is automatically delivered in a device dependent image format and size suitable both for OpenGL ES rendering (see Sec. 3.2) and for tracking.

Tracker. The tracker component implements algorithms for detecting and tracking real world objects from the camera video frames. The results are stored in a state object that is used by the video background renderer and can be accessed from the application code. A single dataset can contain multiple targets. Although the tracker can load multiple datasets, only one can be active at a time. Our system is designed in such a way to scan multiple datasets and perform an automatic dataset switch detection.

Video Background Renderer. The video background renderer component renders the camera image stored in the state object. The performance of the background video rendering is optimized for a wide range of handheld devices.

Our framework initializes the above components and for each processed camera frame, the state object is updated and the render method is called. The framework queries the state object for newly detected targets and updates its logic with the new input data, then it renders the virtual graphics overlay.

As the image targets and the virtual contents (3d models, videos) are loaded in memory, the framework continuously test for the presence of the image targets in the camera field of view. The test is performed at the beginning of each rendering cycle. When the targets are detected and the pose estimated, proper affine transformations such as translation, rotation and scaling are per-

formed in order to correctly render the 3D model and align it to the real environment.

3.2 OpenSceneGraph

OpenSceneGraph [21] is a robust and mature open-source, high-performance 3D graphics toolkit used for developing interactive applications in many different fields like visual simulation, games, virtual reality, scientific visualization and modelling. It is cross-platform and its functionalities are accessible through portable standard C++ code.

OpenSceneGraph uses a scene graph to represent the spatial and logical relationship of the graphical scene. The scene graph is a hierarchical graph not containing direct cycles and isolated nodes, starting from a root node located at the top level. Each node can contain any number of children; entities stored in a node (e.g., 3D models), share common operations and data.

The actual 3D rendering is performed via the OpenGL ES 2.0 API. Such an interface is cross-platform, open-source and designed for embedded systems like handheld devices. Since OpenGL ES 2.0 relies on the use of the programmable pipeline, all the effects, lights, materials and the rendering itself have been managed using shaders.

Although all the rendering setup was realized using OpenSceneGraph, the rendering update is called by Vuforia (see Sec. 3.1), so that framerate is kept synchronized with the speed of pose computation. When the rendering does not involve augmenting reality (i.e., when a single model part is shown), the update function is controlled through a timer object handled by the operating system that allows the application for synchronizing its drawing to the refresh rate of the display.

The OpenSceneGraph module is also responsible for the loading of the 3D models. Usually, a model representing a complex architectural monument is composed by several meshes. These meshes are stored as separate files; at loading time, each mesh is loaded and assigned to a node of the scene graph. All the meshes are loaded concurrently using a secondary thread. Most of the meshes are loaded during the initialization phase when the framework is launched, other ones are loaded on-demand according to the user interaction.

3.3 Structure of the system

The framework is built according to the Model-View-Controller (MVC) design pattern [14]. The management of the application is delegated to a main view controller that initializes all the subcomponents: the Commands view controller, the AR view controller and the OSG view controller.

The Commands view controller is the class devoted to manage all the widgets available in the user interface,

from the buttons to the virtual object selection. This controller captures every command issued by the user and instantiates the execution of an appropriate action, which is often delegated to the appropriate controller. For example, in the case of virtual object picking the computation of which object has been selected is delegated to the OSG view controller. The AR view controller manages the lower part of the augmented reality system. Its main function is to initialize and set up the modules for augmented reality, load the appropriate data set for the targets, and ultimately call the rendering update function, which is then delegated to the OSG view controller. The OSG view controller manages all the rendering calls using the OpenSceneGraph libraries and is responsible for setting up the shaders, loading the 3D object models in memory and managing 3D animations.

The tracking system relies on marker-less feature tracking and vision algorithms to estimate the pose of some predefined image targets. The information regarding such image targets is incorporated in a dataset. Once the dataset is loaded in memory the application is authorized to search for the corresponding targets in the field of view of the camera. After the pose is extracted the 3D models on the scene graph are enabled or disabled based on the application status, then the shader uniforms are updated with the computed model view and projection matrices. For our application we created a custom target image representing a plan of the Imperial Fora area. Such image is designed to be understandable even without the AR layer and to be printed in information boards near the ruins of the Imperial Fora or in AR cards.

4 USER INTERFACE

Given the manipulative interaction style of augmented reality systems, the traditional WIMP (Windows, Icons, Menu, Pointing) interface is not suited for this framework [17]. Rather, the final user interacts with the augmented view by touching the screen of the handheld device. Whenever the framework detects a touch, a line segment intersector is instantiated in order to precisely pick the corresponding virtual object on the screen. The scene graph tree currently rendered by the camera is scanned for intersections. In particular an intersection visitor is executed on the rendering camera, and if one or more intersections are found, then nodes on the path to the first intersection are visited until we find a node labeled for interaction. The selected node is returned to the commands controller which issues a proper action according to the type of object that user touched. Using this fast ray-casting technique, the framework guarantees the interactivity of the user interface even for models composed by a large amount of meshes.

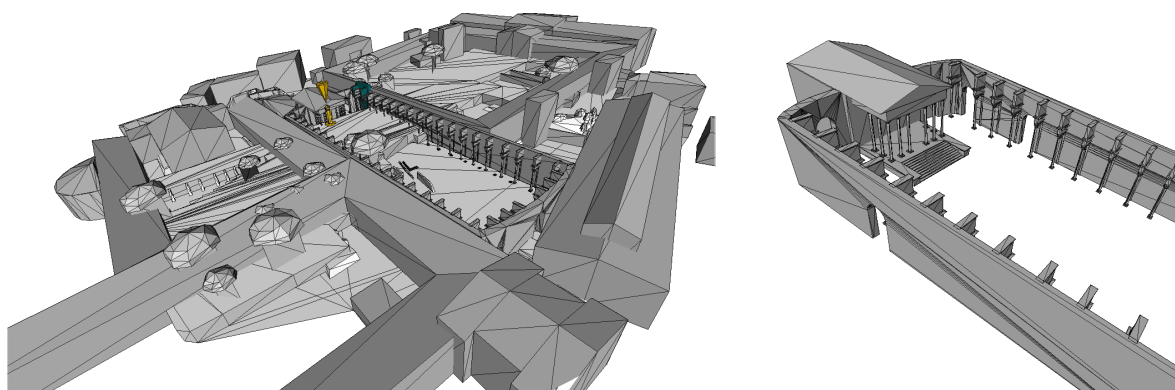


Figure 3: A 3D model representing the Forum of Nerva. It is a set of low-polygonal meshes optimized for interactive rendering and low memory usage.

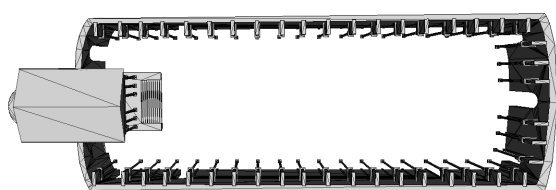


Figure 4: Top view of the 3D representation of the Forum of Nerva.

4.1 3D Visualization

When the user points the camera of the device towards a real area identified by an image target (Sec. 3.1), a 3D virtual model is superimposed to the video feed, providing an alternative representation of the reality. Besides their constant improvement, handheld platforms are in general limited in computational performance and main memory capacity when compared to desktop solutions. Thus, the number of polygons and the related graphics quality detail must be devised in order to be smoothly visualized on a handheld device without sacrificing meaningful details. Fig. 3 and Fig. 4 show the low-polygonal reconstruction of the Forum of Nerva, suitable to be embedded in our system.

For each model, the framework supports the visualization of different historical versions. For example, Fig. 5 shows two different versions of the Forum of Nerva in two different ages, the Imperial Roman age and nowadays.

The 3D model is logically organized in several sensible areas, which can be magnified and interactively manipulated. In this case a new view is presented to the user showing a high-quality rendering of the interested area. The user can zoom in and out the model through a pinch; 3DOFs rotation of the object (roll-pitch-yaw) is performed using an arcball [16] and the scene can be reset to default with a double tap. Fig. 6 shows an example of separate views for details of the Forum of Nerva.

4.2 Virtual Video Streaming

The framework provides the functionality for rendering videos in virtual objects. This is used to create virtual animated buttons and to present informative videos to the user. The videos are visualized beside the 3D model in augmented reality. User can interact with the virtual video in a natural way by tapping for start, pause and resume. Double tapping puts the video in full-screen mode. This has been achieved by exploiting the render-to-texture functionalities of OpenSceneGraph in conjunction with the ffmpeg library, which provides support for many types of video containers and codecs. Fig. 7 shows an example of virtual video applied at the reconstruction of the Forum of Nerva.

5 RESULTS AND CONCLUSIONS

A prototype of the framework has been implemented on top of the iOS operating system; thus it runs on devices such as iPhone, iPad and iPod Touch, that are mainly controlled through the touch interface and are capable of 3D graphics acceleration. However, given the cross-platform nature of all the involved software components, the same framework is easily implementable on other platforms such like Android. The framework achieves interactive rendering of models with over 11000 vertices at 30 frames per second (FPS) on current generation devices mounting a dual core A5 chipset. On previous generation devices (like the iPod Touch 4th generation, single core A4 chipset), the frame rate drops to 5 FPS which can be still considered usable.

The framework allows for the interactive exploration of cultural heritage sites, providing the final users with the possibility to interact with the digital contents in a natural way by using common handheld devices. Through augmented reality techniques, the system provides a solution for the identification of archaeological artifacts and the comparison with their different historical versions.

Our framework allows the visualization of different historical versions of an ancient artifact directly where it

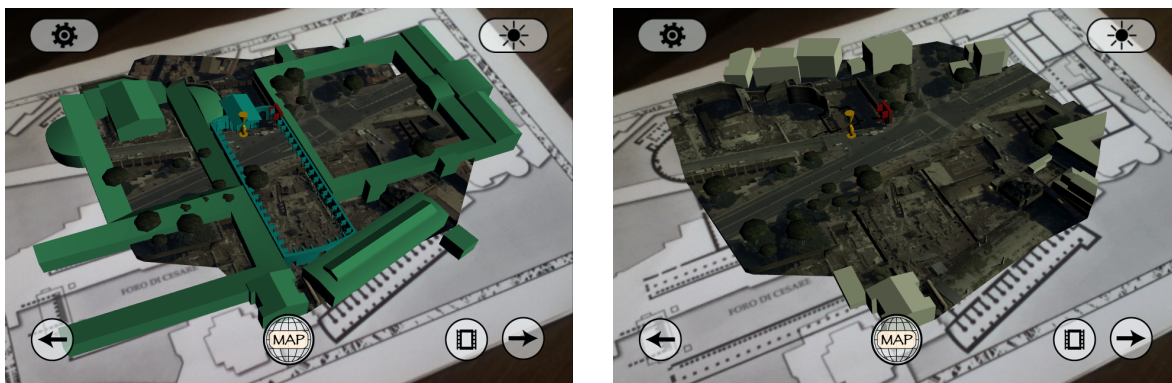


Figure 5: Representation of the Forum of Nerva in (a) the Imperial Roman age and (b) in the current days. Note the red model, namely *Colonnacce*, which is the only remaining part of the forum.

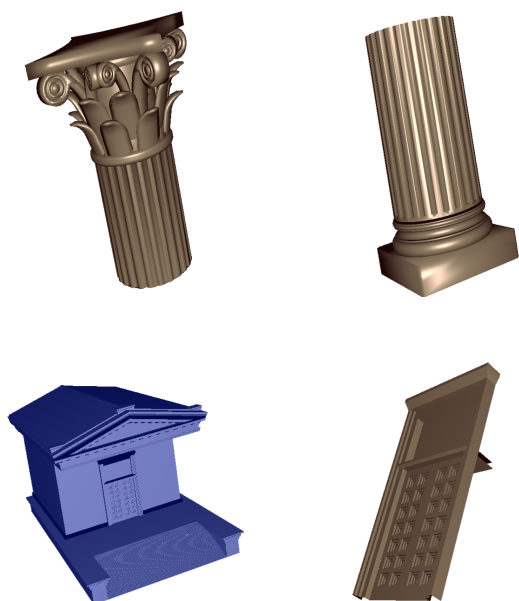


Figure 6: Details of the 3D model can be magnified and explored separately from the augmented view.

was placed originally. The user points the camera of the device towards the on-site ruins' relative target, while the software tracks it and overlays an interactive virtual 3D model of the artifact on it. Some of the most meaningful parts of the model can be selected and magnified to be observed in detail. Special areas of the user interface are devised as 3D *video buttons* embedded into the model. The user can watch the related video together with the 3D model, or in full-screen mode.

The applicability of the framework is tested by providing an augmented view of the Ancient Forum of Nerva. The 3D model has been based and realized according to the documentation provided by historians in the field [18]. The augmented view is superimposed to a sensible map located near the ruins of the ancient forum of Nerva, so that the visitors can observe how

the forum appeared during different historical ages. By tapping on meaningful parts of the model, like the so-called *Colonnacce*, user has access to historical information, audio guide, photo gallery and more about the selected monumental artifacts.

After extracting the pose using an image target at a specific fixed position in the environment, it is possible to project the object at the ground level or in a specific relative position with a simple geometric transformation. However this situation constrains the user to observe the virtual environment without losing the target. Indeed the target loss would cause a lack of information necessary to adjust the relative pose. A possible idea for a future research development may be to compensate this loss of information by estimating camera motion using the data coming from sensors such as GPS, compass and accelerometers, that are usually integrated in handheld devices. In this way user will no more be constrained to maintain the target inside the field of view of the camera.

Plans for future development include further improvement of the user interface and extensive tests on a wider range of datasets (both image targets and 3D representations). Furthermore, we envision this framework to be particularly suitable for the next generation of AR peripherals like the Google Project Glass [2].

6 REFERENCES

- [1] i-Mibac Voyager, March 2013.
- [2] Project Glass, March 2013.
- [3] Rome MVR, March 2013.
- [4] Rome View, March 2013.
- [5] Vuforia SDK, March 2013.
- [6] Ronald Azuma. Tracking requirements for augmented reality. *Commun. ACM*, 36(7):50–51, July 1993.
- [7] Ronald Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355 – 385, August 1997.



Figure 7: Videos embedded into the augmented view.

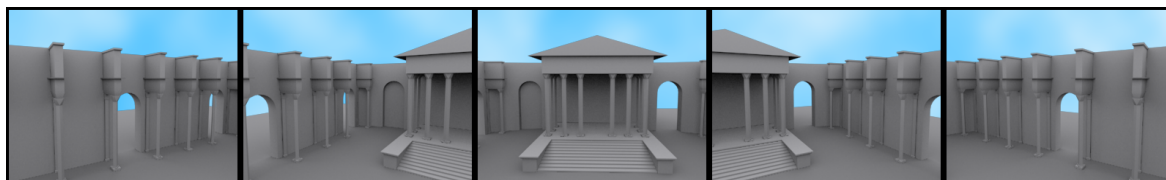


Figure 8: Conceptual rendering example of different camera views, simulating the user turning around inside the Forum of Nerva.

- [8] Erich Bruns, Benjamin Brombach, Thomas Zeidler, and Oliver Bimber. Enabling mobile phones to support large-scale museum guidance. *IEEE MultiMedia*, 14(2):16–25, April 2007.
- [9] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools Appl.*, 51(1):341–377, January 2011.
- [10] Omar Choudary, Vincent Charvillat, Romulus Grigoras, and Pierre Gurdjos. March: mobile augmented reality for cultural heritage. In *Proceedings of the 17th ACM international conference on Multimedia*, MM '09, pages 1023–1024, New York, NY, USA, 2009. ACM.
- [11] P. Dähne and J. Karigiannis. Archeoguide: System architecture of a mobile outdoor augmented reality system. *Mixed and Augmented Reality, IEEE / ACM International Symposium on*, 0:263, 2002.
- [12] Jiang Gao. Hybrid tracking and visual search. In *Proceedings of the 16th ACM international conference on Multimedia*, MM '08, pages 909–912, New York, NY, USA, 2008. ACM.
- [13] Tim Gleue and Patrick Dähne. Design and implementation of a mobile device for outdoor augmented reality in the archeoguide project. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, VAST '01, pages 161–168, New York, NY, USA, 2001. ACM.
- [14] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, August 1988.
- [15] George Papagiannakis, Sébastien Schertenleib, Brian O’Kennedy, Marlene Arevalo-Poizat, Nadia Magnenat-Thalmann, Andrew Stoddart, and Daniel Thalmann. Mixing virtual and real scenes in the site of ancient pompeii: Research articles. *Comput. Animat. Virtual Worlds*, 16(1):11–24, February 2005.
- [16] Ken Shoemake. Graphics gems iv. chapter Arcball rotation control, pages 175–192. Academic Press Professional, Inc., San Diego, CA, USA, 1994.
- [17] Andries van Dam. Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67, February 1997.
- [18] A. Viscogliosi. Il foro transitorio. *Divus Vespasianus. Il bimillenario dei Flavi.*, pages 202–208, 2010.
- [19] Vassilios Vlahakis, Nikolaos Ioannidis, John Karigiannis, Manolis Tsotros, Michael Gounaris, Didier Stricker, Tim Gleue, Patrick Daehne, and Luís Almeida. Archeoguide: An augmented reality guide for archaeological sites. *IEEE Comput. Graph. Appl.*, 22(5):52–60, September 2002.
- [20] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, pages 125–134, Washington, DC, USA, 2008. IEEE Computer Society.
- [21] R. Wang and X. Qian. *Openscenegraph 3.0: Beginner’s Guide*. Packt open source. PACKT PUB, 2010.

A Comparative Analysis of Spatial Partitioning Methods for Large-scale, Real-time Crowd Simulation

Bo Li

University of Canterbury
Christchurch, New Zealand
bli62@uclive.ac.nz

Ramakrishnan Mukundan

University of Canterbury
Christchurch, New Zealand
mukundan@canterbury.ac.nz

ABSTRACT

Acceleration algorithms involving spatial partitioning methods are extensively used in crowd simulation for real-time collision avoidance. Memory and update costs become increasingly important as the crowd size becomes large. The paper presents a detailed analysis of the effectiveness of spatial subdivision data structures, specifically for large-scale crowd simulation. The results demonstrate that a regular grid data structure combined with an extended oriented bounding volume for crowd members can facilitate efficient updates necessary for real-time performance.

Keywords

Crowd simulation, crowd animation, partitioning algorithms, collision detection, subdivision data structures, bounding volumes.

1. INTRODUCTION

Crowd simulation and animation is an active area of research that finds several applications in computer graphics, analysis and design of urban environments and the development of emergency evacuation strategies [SOH11], [WXZ⁺11]. In recent years, real-time crowd simulation applications have gained importance in virtual training systems, such as combat operations training [QC09]. Real-time simulation of large-scale crowd movement requires effective spatial partitioning methods that can provide fast updates.

Space partitioning techniques [Sam06] are widely used for broad phase collision detection between objects and also for collision avoidance with obstacles. The more general problem of crowd detection however addresses various other aspects such as narrow-phase collision detection using intersection tests between pairs of geometrical primitives, and collision response algorithms. Several space partitioning data structures such as quadrees [KLZ08], *k*-d trees [GCL⁺10] and regular grids [BQ10] are commonly used to reduce the number of

comparisons between objects. Bounding interval hierarchies [WK06] are recently introduced data structures that have been found useful in applications such as ray-tracing. A detailed comparative analysis of these data structures in terms of their effectiveness and suitability for large-scale crowd simulation will be useful for the development of real-time applications. This paper presents some of the important results obtained through our research. For convenience, a single member of a crowd is referred to as either a "character" or an "agent." The motion of the crowd is assumed to be confined to a two-dimensional "ground" plane. Thus, even though characters in a crowd and obstacles may have three-dimensional representations, we need consider only a two-dimensional motion projected onto the ground for analysing problems such as collision detection, obstacle avoidance and path planning.

The paper is organised as follows. The next section describes space partitioning data structures considered in our research: Grid, Quadtree, *k*-d tree, and Bounding Interval Hierarchy (BIH). Section 3 presents our crowd simulation model and discusses the implementation aspects of the above four data structures. Results of the comparative analysis are reported in Section 4. Section 5 summarises the main contributions of the paper and outlines future work.

2. SPATIAL PARTITIONING

In this section, four different data structures that are suitable for crowd simulation are discussed, looking

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

at specific advantages and drawbacks of each. Three of the data structures are widely used in broad-phase collision detection and the fourth method [WK06] is

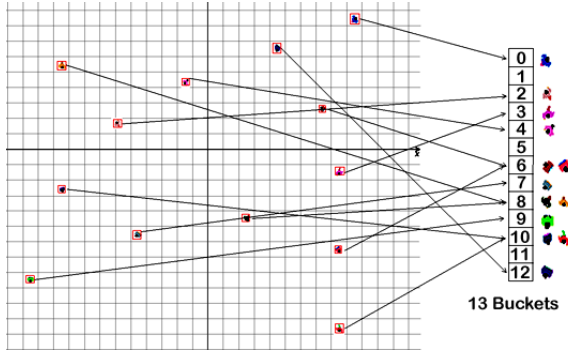


Figure 1. An example of a set of character models in a grid-based partitioning, with a hash table storing object's indices.

used primarily in ray tracing applications. For the purpose of comparison, the brute-force method which compares every pair of character models/agents for collision detection, is also considered in the experimental analysis.

2.1 Regular Grid

A uniform grid [LD08] is a very effective space subdivision scheme. It is fast for collision detection and easy to implement. It partitions the simulation space into small cubic cells with same size. The size of cells is usually defined based on the character's bounding box size, and the character itself is associated with the cell that contains its centre. It is also assumed here that all characters have the same size. Since crowd simulation models will usually consist of large scenes, a regular partitioning of the space into small cells will yield a large number of cells and correspondingly large memory requirement. One straightforward solution is to have spatial hash structures. A spatial hashing [THM⁺03] divides 2D or 3D space into uniform grids, and then uses a hash function to convert them into 1D hashed table. For example, a point with position $p = (x, y, z)$ is hashed into a hash table of size h by computing its cell index c as follows:

$$c = \left(\left(\left(\frac{x}{d} * u \right) \oplus \left(\frac{y}{d} * v \right) \oplus \left(\frac{z}{d} * w \right) \right) \right) \bmod h \quad (1)$$

Where u, v, w are large prime numbers and d is the cell size. If multiple points are hashed to the same hash cell, chaining is employed to resolve these hash collisions, *i.e.*, the points are stored in a linked list specific to this cell. An example is shown in Fig. 1.

An important advantage of the grid based partitioning is that it is easy to build, and the data structure need not be updated any time during the whole simulation.

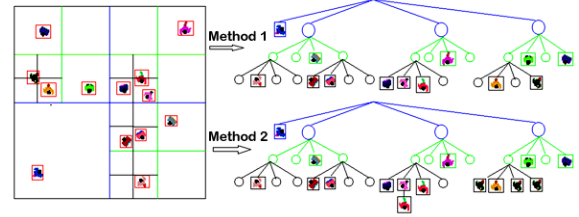


Figure 2. An example of a set of crowd members stored in a Quadtree, showing two storage methods.

In the first method, objects are stored only once based on the location of their centroid, while in the second method, an object is stored in all leaf nodes that intersect the object's bounding volume.

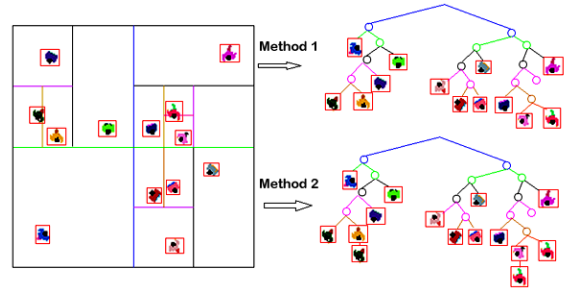


Figure 3. An example of a scene with spatial partitioning using a k -d tree, with two storage methods as in the case of a quadtree.

Further, an efficient hash map provides $O(1)$ search time for finding an object.

2.2 Quadtree

Quadtree is a tree-based partitioning method, and was originally proposed in [FB74]. It was used for processing images and two-dimensional range queries at the early stages, and then found applications in several other areas such as ray tracing and collision detection. It is an axis-aligned hierarchical partitioning of a two dimensional space. Each internal node in the quadtree has exactly four children, and each node also has a finite volume associated with it. A two dimensional world generally is fully enclosed in an axis-aligned bounding square, and is subdivided into four smaller squares at each recursive step (Fig. 2).

Quadtree [PPD07], [ST05] also is a popular acceleration data structure in crowd simulation. It is easily constructed by uniformly subdividing regions containing at least one crowd member into four sub-regions. Compared to the grid, a quadtree generally uses much less memory when members of a crowd are not uniformly distributed in a region. The main

drawback of the quadtree structure is that the tree will need to be rebuilt almost every frame for a highly dynamic scene. The search time for a quadtree is also greater than that of a grid.

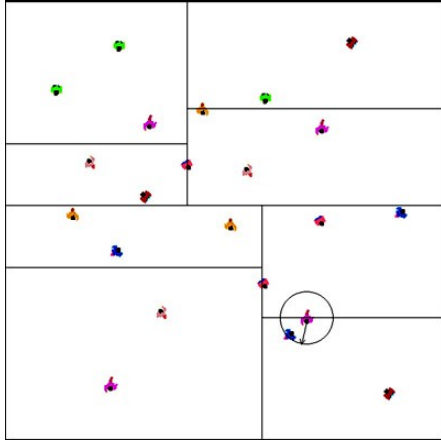


Figure 4. Nearest neighbour search using a k -d tree.

2.3 K -d Tree

A k -d tree is a binary tree, which divides a k -dimensional space hierarchically using a set of axis-aligned splitting planes. It uses a simple construction by dividing a non-empty space and its subspaces using median cut recursively, first using the x -coordinate, then the y -coordinate, and then again using the x -coordinate, and so on (Fig. 3). The depth of the tree is determined such that either the leaf nodes only contain a pre-specified maximum number of objects or the depth of tree has reached a maximum threshold. The algorithm for computing a k -d tree can be optimized by sorting the objects first, and then finding the median objects, and the corresponding splitting planes.

The nearest neighbour search algorithm using a k -d tree effectively finds an agent's neighbours. Given a k -d tree with N nodes, at least $O(\log N)$ inspections are needed on an average, because any nearest neighbour search requires traversal to at least one leaf of the tree. Generally, during a nearest neighbour search, only a few leaf nodes need to be inspected. Fig. 4 shows an example where only two nodes have been visited, the agent with blue background colour is the nearest neighbour for the purple agent.

2.4 Bounding Interval Hierarchy

Bounding interval hierarchies (BIHs) have been recently introduced and used for real-time ray tracing [WK06]. It is found to be faster than k -d trees and easy to implement. BIH is similar to bounding volume hierarchies and k -d trees, and has the advantages of both approaches.

BIH provides very fast construction times and efficient traversal. It uses two parallel partitioning

planes for each node. For a given node, the plane perpendicular to and passing through the midpoint of the longest axis of the node's Axis Aligned Bounding Box (AABB) is first chosen as the splitting plane. Assume that this axis is in the x -direction, and the position of the splitting plane is x_0 . The AABBs of the objects within the node's volume are then sorted along this axis. The objects whose AABBs have all

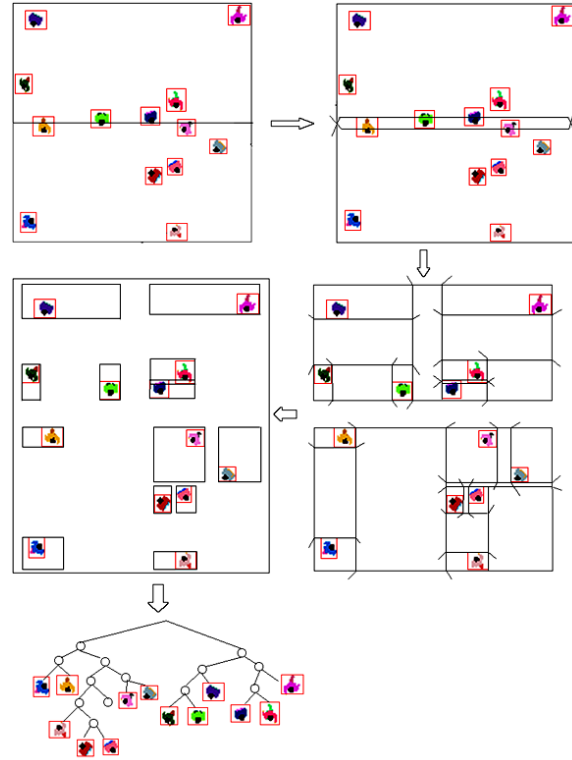


Figure 5. Partitioning of objects into left and right branches using two parallel partitioning planes.

x -coordinates less than or equal to x_0 are assigned to the left child. AABBs that are entirely on the right of the splitting plane are assigned to the right child. Objects whose AABBs intersect the splitting plane are classified as belonging to the left or right child depending on which side of the splitting plane the AABBs have maximum overlap. The left partitioning plane is then defined using the maximum value of the x -coordinates of the AABBs belonging to the left child, and the right plane is defined using the minimum value of the x -coordinates of the AABBs belonging to the right (Fig. 5). The process continues by splitting each child node along the longest axis and defining two partitioning planes along that axis. A node containing only a single object is not subdivided further.

The efficiency of the bounding interval hierarchy is due to the advantages inherited from space partitioning structures similar to a k -d tree. On the

other hand, bounding interval hierarchies have a fixed pre-allocatable size depending on the number of objects. Another advantage inherited from bounding volume hierarchies is that the volume elements can overlap and thus allow efficient update of the structure for dynamic scenes.

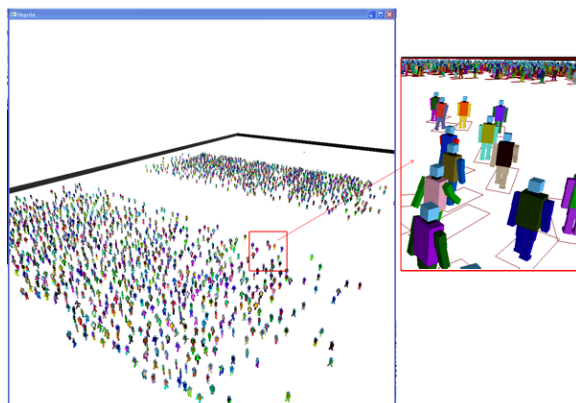


Figure 6. Crowd simulation with 2000 agents, where each agent is bounded by an EOBV of a dynamically changing stride length.

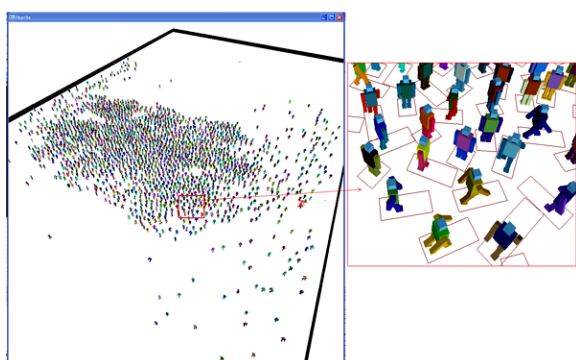


Figure 7. Crowd simulation with 2000 agents, where two groups move towards each other and converge in the middle of the scene.

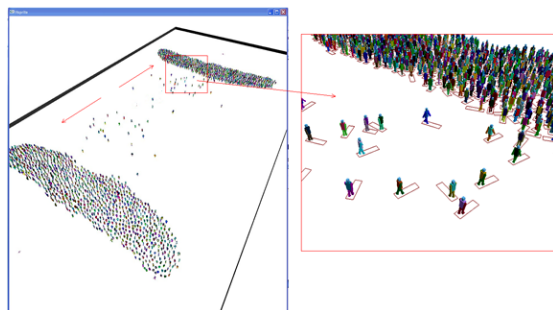


Figure 8: Crowd simulation with 2000 agents, where the whole group moves towards a common destination.

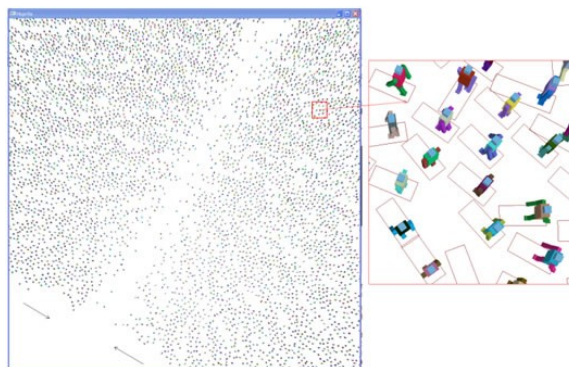


Figure 9. Crowd simulation with 10000 agents, with two groups moving in opposite directions.

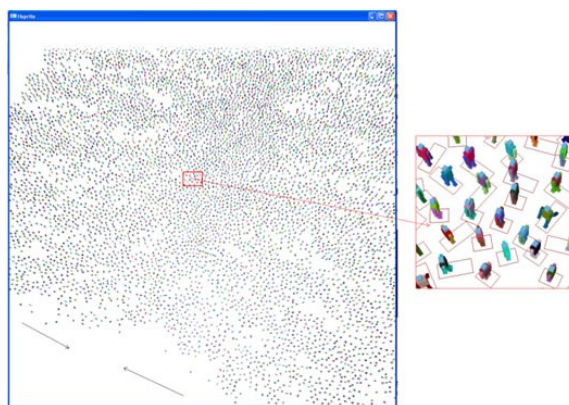


Figure 10. Crowd simulation with 10000 agents, with two large groups merging together in the middle of the scene.

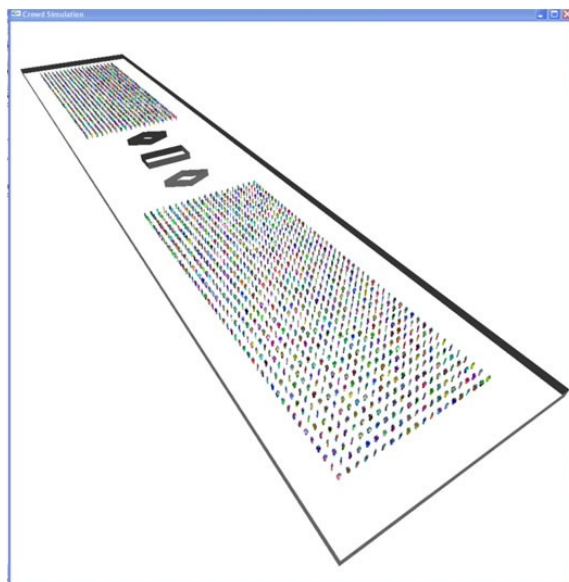


Figure 11. Crowd simulation with 2000 agents with three different obstacles located in middle of the scene.

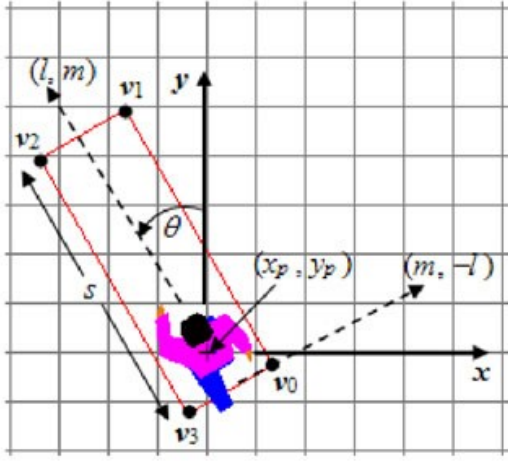


Figure 12. Definition of the extended OBB. [ML12]

3 CROWD SIMULATION MODEL AND IMPLEMENTATION

For the purpose of experimental analysis, we constructed a set of complex 3D scenes (such as Fig. 6-10), each with a large number of crowd members. A typical scenario consisting of two large groups of people moving in opposite directions are shown in Figs. 6-8, with Fig. 6 showing the start configuration of the simulation. The groups meet in the middle of the scene later in the simulation (Fig. 7) and later reach the destination (Fig. 8). In each figure, a small region is enlarged to clearly show the crowd members and their corresponding extended oriented bounding boxes (EOBB) with different stride lengths. We then increase the complexity of the simulation by adding a few obstacles into the scene (Fig. 11). Such a scene can have increasing levels of complexity based on the behaviour models and path planning algorithms used. Poorly designed algorithms can also make inefficient memory requests. We implemented all four data structures described earlier to evaluate the performance of each case with respect to increasing crowd size.

We also used a new data structure called the extended oriented bounding box (EOBB) (Fig. 12) [ML12] into our simulation system. EOBBs are convenient data structures that can be used for both bounding volume and instantaneous motion representation. Using EOBBs, broad-phase collision detection can be performed using OBB overlap tests. The stride length s of an EOBB can be dynamically updated based on the number and positions of character models present in the immediate neighbourhood of an object. EOBBs are also found to be useful for obstacle avoidance and computing path deviations [ML12]. EOBBs were used in all our experiments with the four data structures outlined in the previous section.

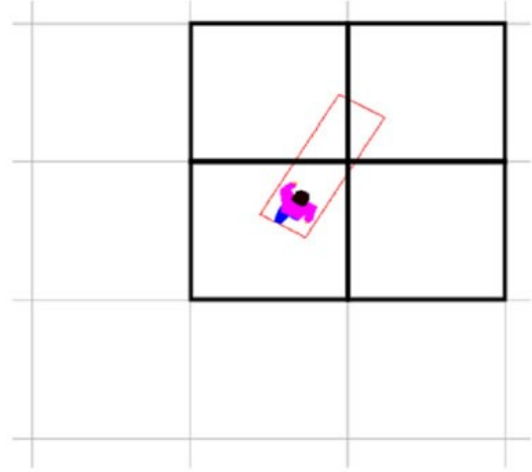


Figure 13. Spatial partitioning using a regular grid where the grid size equals the maximum stride length.

First, we implemented a uniform grid data structure, where the grid size is equal to a predefined maximum stride length. A hash map is used for storing object locations. In [EL07], the authors described a simple and fast way to implement hash functions for minimising the time taken for collision detection. Based on their research, we used a XOR hash function to generate the hash table using Eq. (1), with the values of 73856093 for u and 19349663 for v . While designing the hash table, we need to consider the trade-off between number of cells and memory usage. If we reduce the number of cells, the probability of objects being assigned to the same cell of the hash grid increases. Based on the analysis in [EL07], our hash table has a size h equal to the number of the objects in the crowd simulation.

After the hash spatial data structure is created, the grid neighbour searching method is used for finding all potential colliding agents. The position (x_p, y_p) of a character can be directly used to compute the hash table index as well as the indices of the neighbouring grids. The direction (l, m) is used to select a maximum of three neighbouring cells out of a total of 8 (Fig. 13). If the world coordinate extents of the scene space are given by (x_{min}, y_{min}) , (x_{max}, y_{max}) , and if the maximum stride length used for discretization is s , then the function is given by:

$$k = \left(\frac{x_p - x_{min}}{s} \right) + M \times \left(\frac{y_p - y_{min}}{s} \right) \quad (2)$$

where $M = (x_{max} - x_{min})/s$.

The neighbouring cells are selected based on the signs of the components of the current direction vector (l, m) . For example, the cell vertically above index k given by the index $k+M$ is chosen if $m \geq 0$.

The cell diagonally above the current cell is given by $k+M+1$ is selected if both l and m are positive. Only

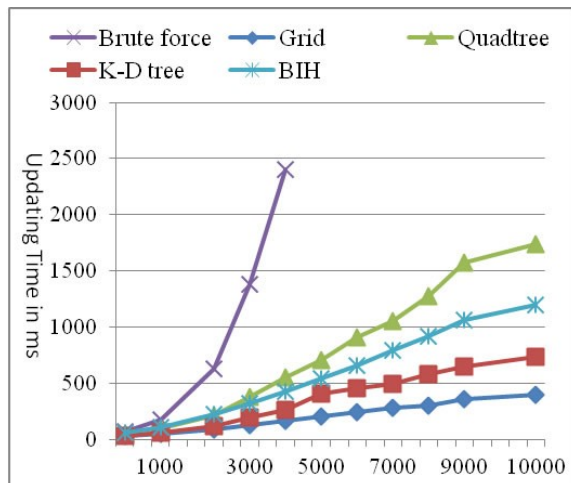


Figure 14. Updating time vs. number of agents (Grid, k -d tree, quadtree, and BIH)

four identified cells (including the current cell) are used for the broad-phase collision detection for each agent.

Second, we implemented a quadtree shown in Fig. 2, and the simulation scene is subdivided into four smaller squares at each recursive step. A leaf node contains only agents. Each agent is stored only once in a leaf node. The centroid of each agent is used to determine which side of the split plane the agent lies. A leaf node size is always larger than the maximum stride length of an agent. For the neighbour search, we first traverse the tree and find the leaf node which contains the agent, then we use OBB-AABB overlap test to find if the agent is fully contained by leaf node. If so, we add all agents in this leaf node to the agent's neighbour. Otherwise, we check which boundary of leaf node is intersected with EOBB of the agent, and then traverse the tree to find the leaf nodes which connect with those boundaries, finally add those agents in the leaf nodes to the agent's neighbours.

Third, a k -d tree is implemented into our system, as shown in Fig. 3. In a highly dynamic scene, the k -d tree will need to be updated almost every frame. By choosing splitting planes properly using median points we can always aim to get a nearly balanced tree. We first choose the longest axis, and use the quick sort algorithm to sort the object coordinates along this axis, and the middle point is chosen as a splitting plane. The goal of this approach is to create subgroups which contain nearly the same amount of objects. Then the recursive spatial partitioning is continued until the depth of the tree has reached a pre-specified number, or if the number of the objects in the leaf nodes is less than a given threshold. A

point on the splitting plane is always stored in left node of the tree. The nearest neighbour search algorithm is used to minimise the number of comparisons in the collision detection phase.

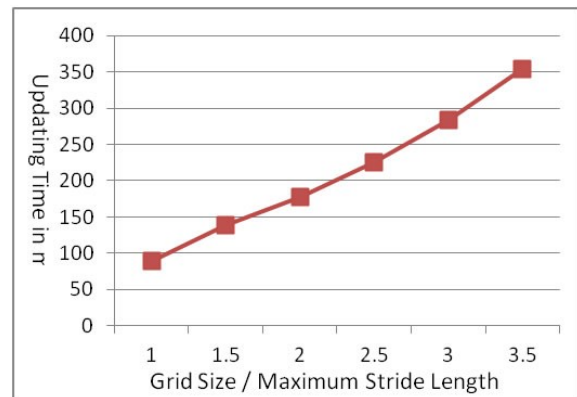


Figure 15. Updating time vs. cell size for the grid structure.

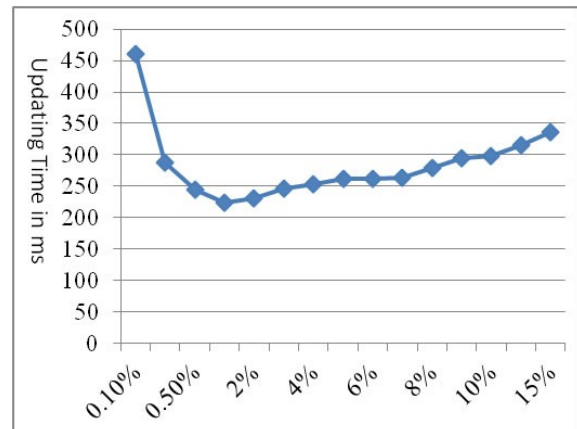


Figure 16. Updating time vs. number of agents in leaf node for a quadtree.

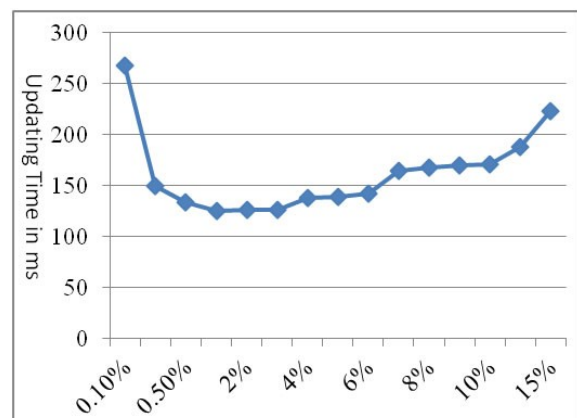


Figure 17. Updating time vs. number of agents in leaf node for a k -d tree.

Finally, we implemented a Bounding Interval Hierarchy for partitioning a crowd scene. The AABB is calculated for each object, the approximate sorting [WK06] is used to sort the agents, and then we determined the two paralleled splitting planes by median-cut. The next section describes experimental results obtained using the above partitioning methods.

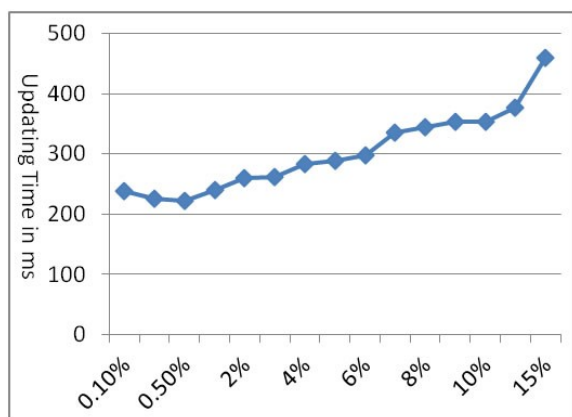


Figure 18. Updating time vs. number of agents in leaf node for a BIH.

4 RESULTS AND EVALUATION

Experimental results shown in this section are generated by simulating different scenarios by increasing crowd size and adding different types of obstacles.

First, we compared the update time for each data structure. The results are depicted in Fig. 14. The graph shows that grid structure gives the best performance. Other types of hierarchical structures required frequent updates because their partitioning algorithms depend on both the position and distribution of crowd members in the constantly changing scene. Even when the number of agents in the scene is 10000, the time taken for updates using the grid structure is less than 500ms. We also noticed that there grid structure does not provide a significant improvement over the brute-force method, when the number of agents is less than 1000.

Cell size is an important parameter in the design of the grid data structure. In our experiment, the size of cell is set up to the maximum stride length of agent first, and then we increased the size of cell, and measured the updating time. The results are shown on Fig. 15. The best performance is obtained when the cell size just fits the maximum stride length of agent.

In Figs. 16, 17, 18, we provide experimental results using quadtree, *k*-d tree, and BIH to find the variation in performance with the maximum number of objects stored in each leaf node. The results show that when

the objects in leaf node is 1% of total number of objects provide the best performance for both quadtree and *k*-d tree. For the BIH, we can subdivide the space until only one object is in the leaf node, and we can still get a good performance, but when the number of objects contained in leaf node is 10, we got the best performance.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a fairly extensive comparative analysis of the performance of spatial subdivision structures in large-scale crowd simulation. The simulation results show that a grid data structure with extended oriented bounding boxes for character models gives the best performance when the number of crowd members is very large. Crowd simulation of 10000 agents in real-time can only be achieved by using such spatial data structures. The grid data structure has the advantage that it doesn't need to be updated, and an efficient hash map implementation can provide fast look-up. The extended oriented bounding box is also found to be very efficient in representing both geometry and instantaneous motion of a character in the crowd.

The paper has presented an overview of four commonly used spatial subdivision methods, and analysis using update time with respect to variations in crowd size, grid cell size, and the maximum number of objects in the leaf nodes of the tree structures for quadtree, *k*-d tree and the bounding interval hierarchy.

Future work in this area is directed towards combining collision avoidance with path/motion planning algorithms incorporating various types of behaviour models. Effective mechanisms for improving the performance of hash mapping for the grid structure will also be explored. A direct extension of the work presented in the paper would be the performance analysis of acceleration algorithms when crowd motion is not confined to a two dimensional plane. Such methods would then heavily rely on multi-dimensional data structures [Sam06] for minimizing comparisons.

When the crowd size increases in scale from large to massive, the performance of acceleration methods becomes crucial. Several models for the simulation and rendering of massive crowds have now been attempted on the GPU [JPZ⁺09], [PJZ⁺08], [PJZ⁺10]. GPU implementations of spatial structures have been successfully tried using just neighbours of agents. Structures similar to the extended oriented bounding boxes could also be explored further, and implemented on parallel architectures.

6 REFERENCES

- [BQ10] F. Bu and C. Qin. Research on the mass events based on grid-agent. Proc. of Youth Conference on Information Computing and Telecommunications, pp. 130–133, 2010.
- [EL07] M. Eitz and G. Lixu. Hierarchical spatial hashing for real-time collision detection. Proc. of IEEE International Conference on Shape Modeling and Applications, pp. 61–70, 2007.
- [FB74] R. A. Finkel and J. L. Bentley. Quadrees: a data structure for retrieval on composite keys. Journal of Acta Informatica, pp. 1–9, 1974.
- [GCL⁺10] S. J. Guy, S. Curtis, M.C. Lin, D. Manocha. PLEdestrians: a least-effort approach to crowd simulation. Proc. of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 119–128, 2010.
- [JPZ⁺09] M. Joselli, E.B. Passos, M. Zamith et. al. A neighbourhood grid data structure for massive 3D crowd simulation on GPU, Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on. IEEE, pp. 121–131, 2009.
- [KLZ08] W. L. Koh, L. Lin, and S. Zhou. Modelling and simulation of pedestrian behaviours,. Proc. of 22nd Workshop on Principles of Advanced and Distributed Simulation, pp. 32–50, 2008.
- [LD08] A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. Computer Graphics Forum, Proc. of the 19th Eurographics Symposium on Rendering, pp. 1235–1244, 2008.
- [ML12] R. Mukundan and B. Li. Crowd simulation: Extended oriented bounding boxes for geometry and motion representation. Proc. of the 27th Conference on Image and Vision Computing New Zealand, pp. 121–125, 2012.
- [PJZ⁺08] E.B. Passos, M. Joselli, M. Zamith, et. al., Supermassive crowd simulation on GPU based on emergent behavior. Proc. of the VII Brazilian Symposium on Computer Games and Digital Entertainment, pp. 70–75, 2008.
- [PJZ⁺10] E.B. Passos, M. Joselli, M. Zamith, et. al. A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU, Computers in Entertainment, Vol. 7, No. 4, Article 60, pp. 1–15, 2009.
- [PPD07] S. Paris, J. Pettré, and S. Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. Proc. of Computer Graphics Forum, pp. 665–674, 2007.
- [QC09] H. Qiu and L. Chen. Real-time virtual military simulation system. Proc. of 1st International Conference on Information Science and Engineering, pp. 1391–1394, 2009.
- [Sam06] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann Publishers, New York, 2006.
- [SOH11] S. Sharma, S. Otunba, and J. Han. Crowd simulation in emergency aircraft evacuation using virtual reality. Proceedings of the 16th International Conference on Computer Games, pp. 12–17, 2011.
- [ST05] W. Shao and D. Terzopoulos. Autonomous pedestrians. Proc. of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 19–28, 2005.
- [THM⁺03] M. Teschner, B. Heidelberger, M. Muller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable models. Proc. of the Vision, Modeling, and Visualization Conference, pp. 19–21, 2003.
- [WK06] C. Wächter and A. Keller. Instant ray tracing: The bounding interval hierarchy. Proc. of the 17th Eurographics conference on Rendering Techniques, pp. 139–149, 2006.
- [WXZ⁺11] X. Wei, M. Xiong, X. Zhang, and D. Chen. A hybrid simulation of large crowd evacuation. Proc. of 17th International Conference on Parallel and Distributed Systems, pp. 971–975, 2011.

High-velocity optical flow

Joris Vergeest

Delft University of Technology

Landbergstraat 15

Delft, The Netherlands

j.s.m.vergeest@tudelft.nl

ABSTRACT

Optical flow is widely used to estimate the velocity of objects relative to a digital camera. Most commonly, two images taken with the same camera at small time difference are compared in order to detect the displacement of structures in 2D image space. Such displacement could be a measure of displacement, or motion, of objects in the scene relative to the camera. At high velocities, the displacement in image space is relatively large and the correlation of image structures gets more difficult. The displacement can be reduced by reducing the time difference, or increasing the number of frames taken per second. However, due to the reduced exposure time, the quality of the individual images gets poorer. In some practical situations, it appears technically very difficult to achieve reliable speed measurement at high velocities, even when using high-speed cameras. One example is the measurement of self-speed from images of the road surface taken with a camera from a driving car. In view of this purpose we explore the potential of the method and its limitations.

Keywords

Optical flow, high speed, noise, low resolution

1. INTRODUCTION

Optical flow is, both in biology and in technology, an important phenomenon. It is the motion of structures in a two-dimensional projection from a three-dimensional scene. In biology, optical flow is considered crucial for animal and human vision, the detection of moving objects and the experience of self-speed. In computer vision, optical flow is the basic observable to quantify speed of objects relative to other objects, or relative to the camera taking the scene. The latter process is referred to as self-speed estimation or measurement.

In this paper we research the feasibility of self-speed measurement of a car driving on a highway based on optical flow, under poor visibility conditions such as low-contrast texture. In section 2 we describe the motivation of the investigation and the requirements of the method. Also the scope of this paper will be precisely defined. In section 3 we present the technical approach and in section 4 we define the experimental conditions. In section 5 we present the experiments and results. Conclusions about the

feasibility of the method and outlook are described in Section 6.

2. MOTIVATION AND REQUIREMENTS

The research aims at a method to extract and record self-speed of a car and in addition visual scene information, using an instrument which can be easily mounted in a car. This paper deals with, and its scope is limited to, the feasibility of such instrument. The background of the research is the study on car drivers' behavior during highway traffic congestions and on new approaches to influence the driving style and to reduce traffic jams. Both the emergence of congestions and their dissolution have got due attention in research over decades. Based on traffic flow models, simulation systems have been developed to study all kind of phenomena of traffic on micro and macro scales, under various conditions in various scenarios. The behavior of individual car drivers is central to most traffic flow models, in particular when they categorize as car-following models. These models assume that a car driver controls his/her car mainly as a function of other cars, in front (if any). The acceleration (positive or negative) of the car is modeled as a function of own speed and the speed of the car ahead and its distance.

Treiber (2003) proposed that car-following characteristics not only differ among drivers, but may also vary for one driver over time. He has modeled

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

memory effects in the response behavior of drivers to the traffic situation, *e.g.* by presuming that after being stuck in a jam, drivers tend to increase the time gap to the preceding vehicle. When incorporating this memory effect into the IDM (Intelligent driver model), traffic flow simulations get increasingly consistent with the measured data (Treiber 2003).

If cars tend to accelerate slowly after leaving a traffic jam, it may cause a significant decrease of road capacity and cause congestions to be over-persistent. Numerical simulations have shown that the life time and length of jams as well as delays of individual cars increase significantly when the acceleration parameters of the IDM are reduced [Vergeest 2012]. Although there are anecdotic indications that people tend to leave traffic jams too slowly, we have not found any objective reporting about this. It is the main goal of our research to obtain statistical information about drivers' behavior just after having been stuck in a jam. Without this information being available, we can only speculate about ways to influence and improve the driving style and thus to avoid OPCs (over-persistent congestions).

We should point out here that the ACC (adaptive cruise control) and similar systems could reduce the problem of OPCs. However, although the penetration rate of ACC is increasing, it is not yet evident that they operate efficiently and safely during congestions and other low-speed situations (Xiong 2012). Therefore we will focus on fully human-controlled cars.

One could reflect about obtaining statistics of human driving behavior by using a car simulator. Suppose that the car simulator were based on the IDM (or similar) model. Then the virtual traffic provided by the simulator accelerates according to the IDM. The subject's car is operated by a test person, allowing the actual acceleration characteristics (and other parameters) of the test person to be recorded. The scenario provided by the simulator may contain congestion conditions. In this way, using the data from many test persons, the acceleration profile (as function of distance and speed) could be statistically obtained. There is, however, one basic assumption undermining this approach. The virtual, surrounding traffic generated by the simulator is based on the IDM and not on the actual acceleration profile, which is actually the unknown we are after. The difference between the model's profile and the real-life profile could bias the profile exhibited by the test persons driving the simulator.

In our aim to obtain the real-life acceleration behavior of drivers we focus on three main parameters: 1) time, 2) the speed of the own vehicle and 3) the distance to the car ahead. From the latter

the speed of the car ahead (relative to the own car) can be derived. Although useful, the location of the own vehicle as a function of time is not needed to detect the occurrence of OPCs. We limit our study to cars in a single lane. Our interest is in situations where cars leave a congestion, which represent, however, only a small fraction of typical journeys.

One way to collect sufficient statistics is to record the three aforementioned parameters of cars participating in traffic over long periods of time. The situations of interest should then be filtered out in subsequent data analysis.

Once somebody volunteers to participate in the research and, it should be made very easy to adapt his/her own car. We formulate the requirements of the data taking system:

1. It should be portable and easily and quickly installable in any common passenger car. The car driver him/herself should be able to install and take out the system in less than 1 minute.
2. It should require no or very little effort or attention to operate the system. A single on/off switch should suffice. Automatic switch on/off is also an option. It should not distract the attention of the driver during driving.
3. The instrument(s) should not be expensive or otherwise attract the attention from people passing by.
4. The recording capacity should be sufficient for about 50 to 100 hours of driving time.

We list no requirements about computer processing, assuming that the data needs not be analyzed real-time. The extraction of the three parameters from the raw data will be done offline as will be the analysis.

Let us define the scope and purpose of this paper more precisely. The main purpose of this paper is to find out whether it is feasible to collect empirical data subject to the 5 requirements listed above. (The conclusion of the paper is that it is not, to our best knowledge). In Section 3 we reason that a possible approach could be based on a simply mountable camera, inside the car, viewing into forward direction. To actually detect optical flow from image sequences can be done with a multitude of methods, as *e.g.* reviewed by [Barron 1994]. Although the performances of the various analysis techniques differ, their effectiveness in view of our application is outside the scope of our paper. The major factor that limits the feasibility seems to be the image quality, rather than analysis performance. Therefore, our main focus is on requirements of image resolution at high self-speed.

3. TECHNICAL APPROACH

The most direct way of recording speed is to simply readout the car's own speedometer. In principle, the speed (as many other data) can be obtained in digital form via a cable connector provided by the car manufacturer. However, the connection and the interface are not standardized, which makes it unpractical, considering requirement 1.

The recording of own speed as a function of time could also be done using a GPS logging device. However, deriving your speed from GPS coordinates and time is sensitive to the accuracy of the GPS coordinates, which is known to vary depending on the quality of and the number of the satellite signals received. Some GPS devices can measure speed directly from the Doppler shift, which is more accurate, but also dependent on the satellite reception quality. In practice, measuring time and own speed using GPS is a good option, and might meet all 4 requirements.

However, we also need a practical way to measure the distance to the car ahead as a function of time. If one would have access to the GPS data from own car and from car ahead, the problem would be solved. In a controlled field experiment, it proved possible to collect data from 50 cars on a real highway, which was reserved for the time of the experiment [Schakel 2010]. Each of the cars was equipped with a cooperative ACC, where the ACCs could communicate among each other. Theoretically, when all cars on all motorways would measure and communicate position, then the distance between any two cars could be obtained, see *e.g.* [Herrara 2010]. In practice this is not yet feasible.

We are therefore looking for a way to measure the distance to the vehicle ahead from our own car, as a function of time. We consider two options. The first is radar-based. This concept is central to ACC systems and works reliably. It conflicts, however, requirement 1 and, since it would measure distance only, the recording of speed would involve one additional device or instrument. The other option is vision. As demonstrated in [Nieto 2010], the detection of own speed and of the distance to cars ahead can be retrieved from footage from a single onboard video camera. The measurement of speed is reliant on the detection of optical flow, or the displacement of image features between two subsequent frames from the video camera. Relatively clear image features are white lane markings on a dark surface road. However, at high velocity, the displacement of lane markings may become as large as the distance between the lane markings themselves, which is complicating the computation of speed [Vergeest 2012a]. Although lane markings are

typically present on highway road surfaces, speed measurement should not depend on their occurrence.

In general, temporal aliasing effects deteriorate the performance of optical flow methods [Marmarella 2012]. However, also poor light conditions, camera noise and motion blur in the individual images pose a problem to the derivation of speed from the images.

However, suppose that we could solve these problems, then we have an instrument that would fulfill all 4 requirements. The instrument is an onboard camera providing images of the scene ahead, at a certain frame rate. Even when the images do not contain a time stamp, knowing the time difference between subsequent images is sufficient for our purpose. From the optical flow we can derive the forward speed of the own car, not necessarily to be determined from lane marks for in case these are absent or too far apart, but in some other way, still to be found. Furthermore, from the same images we can detect the distance of the car ahead, based on Nieto's method or as in [Vergeest 2012a].

We remark that optical flow detection should not necessarily depend on the availability of image frames. If the photo sensors of the camera could be read individually and instantly, the motion of image features might be even better identified.

In conclusion, as a technical approach we consider an onboard camera. However, as the most dominant problem, we need to address self-speed estimation from optical flow at high velocity and poor visibility conditions in absence of clear features such as lane markings.

4. EXPERIMENTAL CONDITIONS

In this section we study the feasibility of estimating self-speed from optical flow. The main principle is to determine the displacement of objects in two subsequent images. We have discarded the rotational component, which can be expected to be small compared to the translational component, at high velocity. Also possible bends or slopes of the road surface are not taken into account, assuming that their effects will be small.

As mentioned, when the scene contains clearly detectable features which can be well correlated, the displacement can be reliably determined, and the speed of the object relative to the camera be estimated [Souhila 2007]. In our application we focus on the road surface. Assuming that (at least locally) the road surface is consistent with a plane parallel to the driving direction of the car, points in the surface can be one-to-one mapped from 3D to points in the image plane of a camera fixed onto the car. Although the optical flow of objects such as trees or buildings may be relatively easily determined in the image

plane, there would be no simple one-to-one mapping to the 3D scene.

When the road surface does not contain obvious features, such as lane markings, the visual structure or texture of the road surface might be used to detect the amount of shift as a function of time as a measure of optical flow. When two images of the road surface are available, taken from the same camera at a small time difference, the matching criterion can be defined as a difference function of the two images, which should be minimized [Mammarella 2012]. At low speed, when the amount of shift is small, it is

relatively easy to find matching regions in the two images. At high speed, there are two problems of finding matching image regions. The first is that the expected shift will be large, and therefore a larger range of potential shift needs to be considered. The probability that nearly similar structures pop up will increase, and so will the risk of false matches. Second, due to the high speed the image quality will degrade, either due to motion blur, to shorter shutter times (and hence an increased camera noise level), or both. When reducing the first problem (increasing the frame rate) the second problem gets worse.

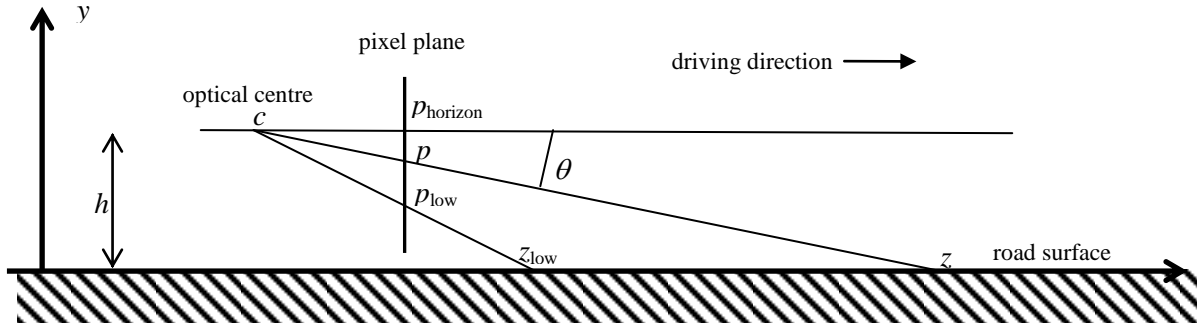


Figure 1. Perspective parameters of the camera setup. The optical center c is at height h above the road surface. The driving direction is into the z -direction.

Let's consider the simplified camera setup in Figure 1. The camera is mounted inside the car near the front window, pointing into the driving direction, taking images as in Figure 2. In the pixel plane of the camera, which is assumed to be vertical, $p_{horizon}$ is the index (counting from bottom up) of the pixel line representing the horizon. p_{low} is the lowest pixel line showing road surface. That particular line in the road surface has z -coordinate z_{low} as measured in the coordinate frame with origin c . The height of the optical center c has y -coordinate equal to h . A point in the road surface at distance z from the camera in forward direction will be mapped to a pixel on scan line p , such that

$$z = z_{low} \frac{p_{horizon} - p_{low}}{p_{horizon} - p}. \quad (1)$$

We define Δp as the amount of vertical shift observed for a point on the surface between two successive camera pictures. Δp depends on the speed v of the car, the resolution and the frame rate of the camera, and on the location of the point in the perspective image. For points projected near the bottom of the picture, the optical flow measured in pixel shift is relatively large. For large Δp the image matching process will be computationally more involved and the risk of error will increase. Therefore, Δp will be an important parameter for the trade-off between image quality and maximum speed.



Figure 2. Picture taken with an onboard camera. The region of analysis is indicated by the white box below the center of the image.

In the exploration of the feasibility of image matching, we will set some further limitations to our scope. Out of the various types of difference functions we chose to use the sum of absolute differences (SAD) among pixel brightness of the image regions. Another assumption is that the image regions are located near the plane $x=0$ of the camera, as to simplify the compensation for perspective distortion.

The focus of initial experimentation will be the ability to determine Δp from two given images, where we assume that 1) the images have been taken by the same camera, 2) a predefined region in the image is of interest, 3) the image in the region of

interest is created according to the setup as in Figure 1.

5. EXPERIMENTAL RESULTS

We collected images with an HD-HERO2 video camera from GoPro [Gopro 2013]. This camera can easily be installed as a dashboard camera. We setup the camera at frame rate $u = 120$ fps. The field of view was 170° at a resolution of 848×480 pixels. The optic flow of the road surface was determined from the pixel data in a small region of the images near $z = 4.2$ m, see Figure 2. In this particular case the region is of size 180×32 pixels. The SAD is determined by moving a subwindow (smaller than the region) from one image over the region of the second image in steps of one pixel. In the current algorithm we ignore the (small) perspective distortion present in the regions. Suppose that we choose the subwindow to have size 160×12 pixels. Then the maximum number of steps in the Y-direction is $32 - 12 = 20$, that is 9 pixels in each direction, which is the maximum Δp that would be detectable.

For the camera setup we have $z_{low} = 4.0$ m, $p_{low} = 150$ and $p_{hor} = 320$. The mapping factor f is the distance on road surface corresponding to an increment of one scan line,

$$f = \frac{\partial z}{\partial p} = \frac{z^2}{z_{low}(p_{hor} - p_{low})}, \quad (2)$$

as can be derived from equation (1). At the bottom of the image region, which is near $z = z_{low}$, a shift of one pixel into the vertical direction corresponds with approximately $f = 4.0/(320 - 150) = 0.024$ m distance on the road surface for our specific camera setting. Since $\Delta p = v / uf$, we have $\Delta p = 3.5$ pixels at $v = 10$ m/s, or $\Delta p = 11.6$ pixels at 120 km/h.

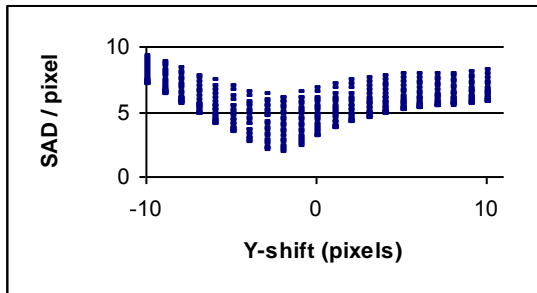


Figure 3. SAD of two consecutive images as a function of Y-shift. For each Y-shift 21 points are shown for X-shift = -10, -9, ..., 9, 10 pixels.

Figure 3 shows, as an example for one pair of images, the SAD (divided by the number of pixels in the sub window) as a function of Y-shift. In this case we observe a minimal SAD at Y-shift or $\Delta p \approx -2$, corresponding to $v \approx 6$ m/s or 21 km/h. The lowest

point in the plot of Figure 3 corresponds to an X-shift of 0 (not visible in the plot). The speed derived from the plot is $v = u f \Delta p$, that is it is therefore discrete in steps of $u f = 2.9$ m/s or 10.4 km/h. One could attempt to fit a curve to the minimum SAD as a function of Y-shift and thus estimate v ; we have not done that.

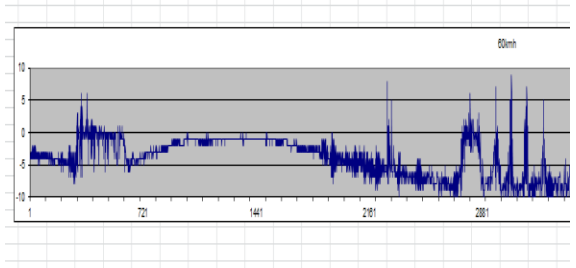


Figure 4. Δp as a function of time over a 30s time interval.

The footage from the HERO2 allows the measurement of v at its frame rate, provided that we can determine Δp . An impression of the reliability to determine Δp is given in Figure 4. The course of Δp can be recognized, but there are some “noisy” parts, which correspond to locations on the road where the surface lacks visual contrast. Figure 5 provides more detail about this effect.

In Figure 5 we show the measured speed as a function of time over a 1 second time interval. The hosting car was driving at a constant speed of approximately 30 km/h. In the plot the speed is presented in units of Δp . The SAD and the mean AD (the absolute differences averaged over all computed shifts) are included in the figure as well, where $AD \geq SAD$. For $10.4 < t < 11.1$ s Δp takes the value -2 , which is quite consistent with the actual speed. Outside the interval, Δp seems scattered. Where both AD and SAD are small, Δp cannot be determined reliably; the image pairs do not exhibit enough contrast. Figure 6 depicts the scene where the change of road surface texture from rough to smooth occurs, near $t = 11.1$ s.

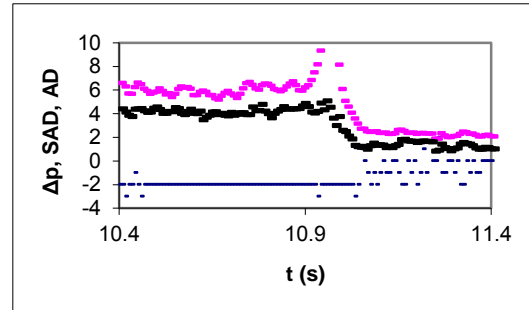


Figure 5. Δp , SAD and mean AD as a function of time, over a one second time interval.



Figure 6. Road surface texture changes from high to low contrast at $t = 11.1s$.

When a smooth road surface exhibits little contrast, the SAD as function of Y -shift (Figure 7) is very different from the profile obtained in Figure 3. In Figure 7 we notice that both the SAD values themselves as their variation due to shift are much smaller.

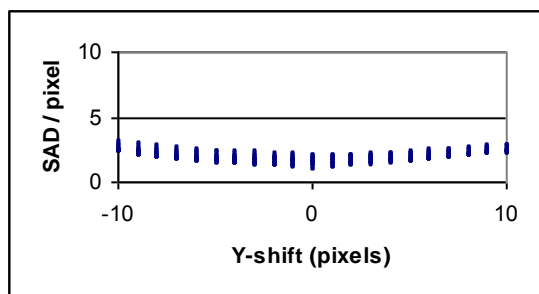


Figure 7. SAD for an image pair of low contrast road surface. For convenience of comparison, the plot scales are the same as for Figure 3.

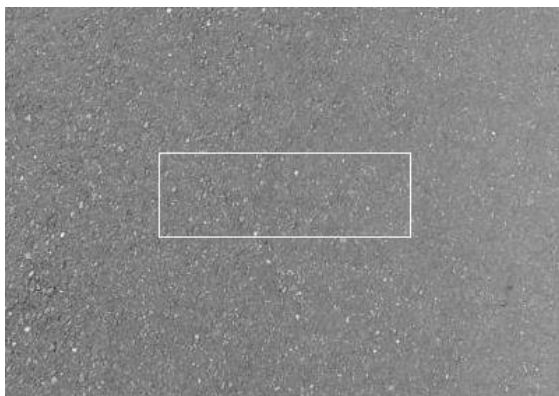


Figure 8. Part of the low-contrast road surface (also visible in Figure 6), taken at higher resolution. The dimensions of the view are approximately 40×45 cm. The white box represents the portion of the image which is used for view matching.

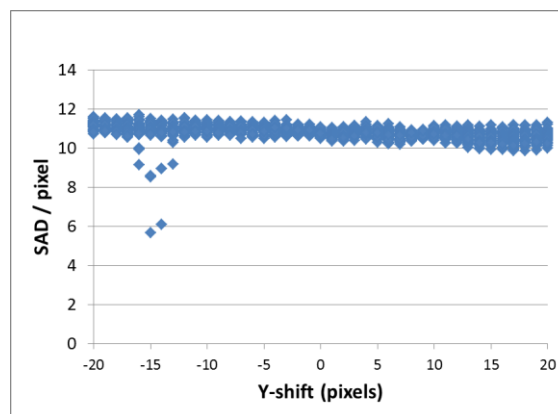


Figure 9. SAD for an image pair of low contrast surface, where images are taken at high resolution.

As mentioned, there are several parameters that may influence the SAD profile, such as the frame rate of the camera, its resolution and shutter time, but also parameters of the software including the choice and size of image region and subwindow.

Concerning the image contrast itself, for now we consider two criteria. First, the degree of contrast that is required to achieve matching. Second, the degradation of contrast due to the speed of optical flow.

With a photo camera we took a detailed still picture of the road surface in the low-contrast region, see Figure 8. Another similar picture was taken after the camera was manually repositioned a few centimeter further in the positive z -direction. Whereas the particular part of the surface road did not show contrast in Figure 6, it does in Figure 8.

From two detailed images we could reliably find a match near $\Delta p = -15$, see Figure 9. The Y -shift in Figure 9 maps to a speed relative to the road surface as by equation (2), where f differs from the mapping factor we applied so far since Figure 9 has been obtained with a different camera setup.

We thus found that the speed of the car could be determined from detailed pictures as in Figure 8, even when the image contrast is low. However, the picture pair from which Figure 9 is derived would not be reproducible with the video camera we applied earlier, for two reasons. First, due to the high resolution, the optic flow measured in pixels/s would be very high (in the order of 3000 pixels/s) and hence a very short shutter time would be required to obtain a non-blurred image. Second, the frame rate should be an order of magnitude larger, to keep the Y -shift between the two images between 10 and 20 pixels. This latter requirement is of course depending on the maximum speed we intend to measure.

6. CONCLUSIONS

If a road surface exhibits sufficient visual contrast, the optical flow can be captured with simple equipment at low cost. The collection of large statistics data over long traveling times about the driving speed of a car (and its distance to cars in front) would then be feasible. We have demonstrated that the footage from a simple video camera is sufficient to measure the car's speed without being dependent on artificial features such as white road markings.

When the surface road contrast gets low and/or the car speed gets high, the method becomes unreliable. However, even low-contrast asphalt exhibits texture from which optical flow can be detected, provided that high-quality images at small time intervals were available. It would not be necessary to continuously store images at a high frame rate. For our experimental research it is sufficient to obtain (for example) only one image pair per second, where the time difference between the images is small (perhaps in the order of 0.1ms or less). We have not yet found a device which could perform like this.

A possible method could be to apply two photo cameras. The cameras should be mounted closely adjacent, aiming at the same point on the road, probably highly zoomed. Shutter times should be short. Then the cameras should be triggered to take one picture every second, where one camera is triggered 0.1ms later than the other one. If pictures as in Figure 7 are obtained this way, the optical flow and hence the speed of the car relative to the ground can be recorded as a function of time.

There is another advantage of applying higher resolution, and thus small f , considering the definition of f in equation (2). The speed is derived from the Y -shift, which is essentially a discrete value, although the appearance of plots as in Figure 3 suggest that a curve can be fitted against the data, from which a minimum could be derived. If the discrete minimum is used, the speed $v = u f \Delta p$ is measured in steps of uf . Since the frame rate u should be high, f should be as small as possible in order to determine v accurately.

Another factor is the luminous intensity of the scene. If high intensity spotlights are applied, a small f can be reached. It would involve the installing of extra high-power lights on a car, which violates requirement 1.

As mentioned, the detection and quantification of optic flow does not necessarily require image frames. If individual pixels of the optical sensor could be read-out at high speed, for example 10KHz, then even small and low-contrast features might be traceable. Their speed in sensor space

would be a measure of the car's speed. A similar principle is applied in the optical mouse.

Another option could be the projection of a laser beam onto the road surface during a predefined time period x . A picture of the road surface taken with a relatively long lens opening time (much longer than x seconds) will show a line on the road surface caused by the laser beam. The length of the line, which can be reconstructed using equation (2) is proportional to v . The advantage of this method is that low-quality cameras could be applied. However, the total setup of equipment gets more complicated due to the inclusion of the laser device.

At present, the most feasible method (fulfilling the 4 requirements) seems to be based on optical flow detection using a single camera or a synchronized pair of cameras. It is still an open question whether images with quality comparable to the one in Figure 8, can be obtained from a car driving at high speed on a road with low-contrast surface texture. Weather and light conditions play an important role as well.

7. REFERENCES

- [Barron 1009] J.L. Barron, D.J. Fleet, S.S. Beauchemin, Performance of Optical Flow Techniques. *Int. Journal of Computer Vision*, 12:1, pp 43-77.
- [Gopro 2013] www.gorpo.com.
- [Herrera 2010] D.B. Work, R. Herring, X. Ban, Q. Jacobson, A.M. Bayen, Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. *Transportation Research Part C* (2010), pp 568-583.
- [Mammarella 2012] M. Mammarella, G. Campa, M.L. Fravolini, M.R. Napolitano, Comparing optical flow algorithms using 6-DOF motion of real-world rigid objects. *IEEE Trans on Systems, Man and Cybernetics – Part C*, Vol. 42, No 6, pp 1752-1762.
- [Nieto 2010] M. Nieto, J. Arróspide Laborda, L. Salgado (2010), "Road environment modeling using robust perspective analysis and recursive Bayesian segmentation". *Machine Vision and Applications*, 7 August 2010.
- [Schakel 2010] W.J. Schakel, B. van Arem, Effects of cooperative adaptive cruise control on traffic flow stability. *Proc. 13th Int. IEEE Annual Conference on Intelligent Transportation Systems*, pp 759-764.
- [Souhila 2007] K. Souhila and A. Karim, "Optical Flow based robot obstacle avoidance," *Int. J.*

Adv. Robot. Syst., vol. 4, no. 1, pp. 13–16, 2007.

[Treiber 2003] M. Treiber and D. Helbing, Memory effects in microscopic traffic models and wide scattering in flow-density data. *Physical Review Letters* E 68, 046119 (2003), pp 1-8.

[Vergeest 2012] J.S.M. Vergeest and B. van Arem, The effect of vehicle acceleration near traffic congestion fronts. In *2012 IEEE Intelligent Vehicles Symposium IV'12*, pp. 45-50.

[Vergeest 2012a] J.S.M. Vergeest, Self-speed and headway measurement in highway traffic from onboard video footage. In V. Skala *et al.* (Eds.) *Proc. of the WSCG 2012 Conference*.

[Xiong 2012] H. Xiong and L. Ng Boyle, J. Moeckli, B.R. Dow and T.L. Brown, Use patterns among early adopters of adaptive cruise control. *Human Factors*, Vol. 54, No. 5, October 2012, pp.722-733.

Efficient Removal of Inconsistencies in Large Multi-Scan Point Clouds

Thomas Kanzok¹

thomas.kanzok

Falk Süß¹

Lars Linsen²

l.linsen

Paul Rosenthal¹

paul.rosenthal

¹ Chemnitz University of Technology
Department of Computer Science
Visual Computing Laboratory
Straße der Nationen 62
09111 Chemnitz, Germany
[e.mail]@informatik.tu-chemnitz.de

² Jacobs University
School of Engineering & Science
Visualization and Computer Graphics Laboratory
Campus Ring 1
28759 Bremen, Germany
[e.mail]@jacobs-university.de

ABSTRACT

When large scale structures are to be digitized using laser scanning, usually multiple scans have to be registered and merged to cover the whole scene. During the scanning process movement in the scene and equipment standing around – which is not always avoidable – may induce artifacts. Since the scans often overlap considerably another scan of the same area might be artifact-free. In this paper we describe an approach to find these "temporal" artifacts (so-called, because they only appear during one single scan) based on shadow mapping, which makes it implementable as a shader program and therefore very fast. The effectiveness of the approach is demonstrated with the help of large-scale real-world data.

Keywords

Outlier Removal, Point Cloud Processing, Laser Scanning

1 INTRODUCTION

3D-scanning has gained increasing popularity in a wide range of applications, including content creation for games and movies [ARL⁺10], reverse engineering [IM09] and acquisition of geometric data for documentation of archaeological sites [BGM⁺09, GSS08, LNCV10] and large-scale structures [PV09, SZW09, WBB⁺08], where new rendering approaches [KLR12, DRL10] have already helped to reduce the postprocessing time for a dataset. In these applications, usually several scans have to be done and registered in order to capture a complete site or building. During this process it is not always possible to completely close the site – or street, in our particular case – for traffic or visitors, or to ensure that no people or equipment of the scanning team are present in the scene.

This leads to ghost geometry that is only captured by one or few scanners and is sometimes not even consistently colored, since colors are acquired independently

from the geometry by different terrestrial laser scanning systems in a second scanning pass. This means that moving objects – which we call "temporary" for the remainder of this paper, because they are not persistent in the scene – introduce artifacts in the scan that compromise the rendering quality considerably (see Figure 1).

A simple observation we made, was that the artifacts induced by temporary geometry are in most cases only present in one of the many merged scans. Since the scans have to overlap significantly to allow robust registration, there is usually at least one scanner that can literally see through such artifacts. The question we have to ask in order to decide whether a point can be considered an artifact is therefore: "Is this point invisible for any other scanner that could potentially see it?" or shorter "Can any other scanner see through this point?"

The second formulation is equivalent to the question whether this point casts a shadow in all other scans that cover it. This suggests applying some variation of shadow mapping to the problem.

In this paper we present an implementation of a shadow-mapping based algorithm for artifact removal in preregistered point cloud datasets. Since all calculations can be done in a shader program on the GPU we can show that the running time of the algorithm is only bounded by the transfer speed of the storage device that holds the data. To achieve this we have to sacrifice some precision, because we can not hold all available

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

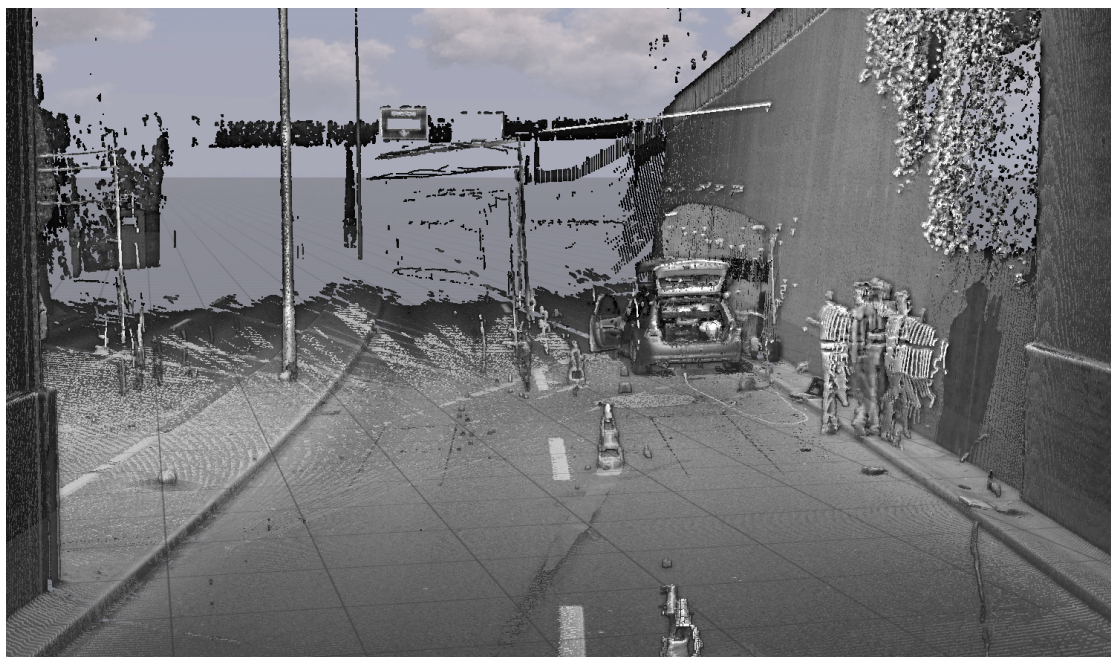


Figure 1: An example for typical temporal artifacts in a scene that was merged from multiple scans. The people in the foreground and the car in the background were only present during one scan. Additionally, there are lots of stripes induced by passing trucks. The scene was shaded to enhance geometry perception.

information on the GPU. This is only critical in the vicinity of edges, however, and can in most cases be treated by detecting these edges in the image.

2 RELATED WORK

Artifacts or outliers are a problem every automated scanning system has to deal with. There are basically two causes for them to occur: inaccuracies in the scanning equipment itself and inconsistencies in the scene that is scanned. Most published approaches do not differentiate between the two sources, so every kind of artifact is treated in the same way and is often characterized as diverging from a statistical distribution. This way the local consistency of the data with a fitted model can be evaluated to identify outliers.

Papadimitriou et al. [PKG03] detect outliers by comparing the local sampling density of a point to the average local sampling density of its neighbors in a certain radius. A similar approach is used by Sotoodeh [Sot06] to define a *local outlier factor* that is based on the relative local size of the k -neighborhood of a point. Another algorithm based on local coherence was presented by Weyrich et al. [WPK⁺04]. They define three criteria for outlier classification: the divergence from a fitting plane in the k -neighborhood, the distance of a point to a ball fitted to its k -neighborhood (without the point itself), and the "nearest neighbor reciprocity", which represents the number of nearest neighbors of a point \mathbf{p} that do *not* have \mathbf{p} as a nearest neighbor themselves.

Large scale architectural datasets are mostly comprised of planar elements (walls, floors, etc.), outliers can

therefore be assumed to form lines or uncorrelated clusters [WBB⁺08]. Artifacts can then be found by analyzing the eigenvalues of a point's covariance matrix. Schall et al. [SBS05] use a kernel density estimation scheme to identify outliers in the data as those points, that have a lower local density than the rest of the data, according to an appropriately chosen threshold.

Directly on the scanned range image operates the method of Rusinkiewicz et al. [RHHL02]. They triangulate the range images acquired by the scanner, reject triangles that have exceptionally long edges or are facing away from the scanner and finally delete single points that are not part of any triangle. Other approaches do not remove outliers at all but try to re-integrate them into the surface by some kind of weighted smoothing scheme [MT09, PMG04].

However, these approaches do not take into account that we actually have much more information than raw point coordinates and possibly color. As already pointed out by Köhler et al. [KNRS12] they therefore fail to recognize ghost geometry, because it is consistent with a model – even though temporal – and tend to smooth close outliers into the surface. Their alternative approach takes an additional camera in a structured-light system for surface recognition and uses the light projector for correcting the measured positions.

In the following paper we show another property of real world data that can be used to detect artifacts – the consistency of valid geometry over multiple partial scans of a large scene.

3 GENERAL APPROACH

In this paper we make a clear distinction between "artifacts" and "outliers". While our definition of outliers follows the conventional concept as being points that do not belong to the model according to some local consistency criterion, we denote as *artifacts* points or larger groups of points that are not part of the global model, although they can be perfectly consistent for one single scan (meaning that they are not technically outliers for this scan), since they may represent temporary geometry. Artifacts therefore include all outliers, but also encompass all temporary geometry in the scene.

The property of merged point clouds we want to exploit to identify these artifacts is illustrated in Figure 2. It is based on the simple observation that "permanent" objects, i.e. objects that are not movable and should therefore appear at the same position in each scan, block the laser beams from any scanner involved. This means, if we compute a shadow map for each distinct scanner using only its "own" points, permanent objects cast a shadow in each of these maps. In contrast, "temporary" objects, i.e. objects that moved during scans, only cast a shadow in one or few scans. We can use this to decide whether a point of the final scan is temporary by checking it against each of the computed shadow maps. If the point is shadowed in *all* shadow maps, it is likely to be permanent, otherwise, if it is *not* shadowed in one or more shadow maps, it is most likely temporary. In the following we describe in detail how we come from the raw scanner output to a classification of each point and show results and timings we achieved with the method.

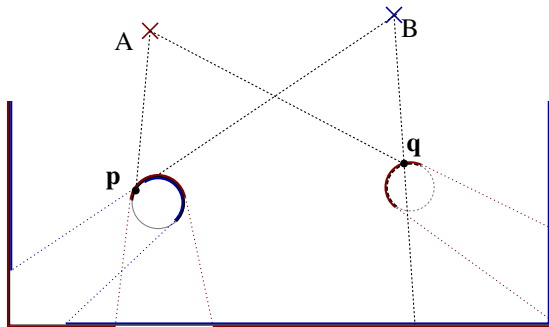


Figure 2: A 2D illustration of the reasoning behind the algorithm. The distance of p to B is larger or equal than the respective depth buffer value stored for B , while the distance of q (which was only seen by scanner A) to B is smaller than the respective depth buffer value stored for B . This means that q has not been seen when scanning from B , while p has been seen (unless it is occluded by a point that is even closer to B). Consequently, q must have been a removable object, while p is a steady object.

Large scale laser scanning systems typically use one laser beam that is reflected by a movable mirror into a spherical direction θ, φ , with θ being the azimuthal and

φ being the polar angle. From the time it takes the laser signal to be reflected back to the scanner the spherical distance r can be calculated, which gives the spherical coordinates $\mathbf{p}_s = (r, \theta, \varphi)$ for the point. They then get converted to Euclidean coordinates $\mathbf{p}_e = (x, y, z)$, which is the output of the scanner.

Different scans in one scene then have to be registered into one global system using strategically placed markers and/or local optimization methods, e.g. multiple variants of the well-known ICP algorithm [CM92]. For this work we assume that this registration has already been done as exact as possible and will not go into further detail here.

In order to identify points that are transparent for at least one scanner, we employ a map that facilitates occlusion lookups. Analogous to the original shadow-mapping algorithm we allocate a texture for each scanner that stores the distances from the scanner to the surrounding geometry. Note that this only conveys useful information if we generate the shadow map for each scanner only on the data that was actually acquired by this scanner.

Usually, the numbers of discrete steps taken for θ and φ are fixed and independent, leading to a uniform sampling in θ and φ . To preserve this uniformity and to lose as little information as possible to sampling errors, we store all information for a scanner in one single texture that is parameterized over θ and φ .

The scanner's sampling rate of k steps in θ and l steps in φ then gives us a very good estimate for the optimal resolution of a spherical shadow map. Using a map with domain $[0, k] \times [0, l]$ would ensure that on average each pixel is hit by exactly one laser beam. In practice, this would also mean that we would have to store one third of the size of the entire dataset on the GPU (one float for the spherical radius r instead of three floats for (x, y, z)), which is normally not possible. Additionally, if the sampling grid is not given for the dataset, we have to reconstruct θ and φ from the Euclidean coordinates, which introduces considerable jitter leading to holes in the map. Due to these limitations, the maximum resolution for the shadow map on the GPU should not exceed $\frac{k}{2} \times \frac{l}{2}$ to ensure closed surfaces and should probably be even smaller ($\frac{k}{f} \times \frac{l}{f}; f \geq 2$, where f is chosen such that the maps of every scanner fit in GPU memory).

3.1 Removing Inconsistencies with Shadow Mapping

If the sampling grid of the scanner is not given in advance, each point \mathbf{p}_e of a single scan in Euclidean coordinates can be converted back to spherical coordinates in a shader program and its r -component can be rendered into a shadow texture image S of size $\frac{k}{f} \times \frac{l}{f}$:

$$S(\hat{\theta}, \hat{\phi}) = r \quad \text{with} \quad \hat{\theta} = \frac{k\theta}{2f\pi} \quad \text{and} \quad \hat{\phi} = \frac{l\varphi}{f\pi};$$

with r being the Euclidean distance of \mathbf{p}_e to the respective scanner origin and $\hat{\theta}, \hat{\phi}$ being the mapping of polar and azimuthal angle to texture coordinates of the render target.



Figure 3: The color texture for the spherical surrounding area of one example scan. The shadow map created from this scan can be seen in Figure 7a.

Having computed a shadow map for each of the n scanners (see Figure 3 for an example of a scanned environment) the shadow maps are store in a layered 3D texture. Using this texture in connection with the original scanner positions, the complete registered point cloud is processed by a vertex shader program, that executes Algorithm 1 and returns its result via transform feedback.

In practice we have to make several tweaks to the algorithm in order to account for different problems which we describe in detail in the following chapters.

3.2 Handling Anisotropy

The first problem is caused by the fact that we will almost never have a spherical scene around a scanner. On the contrary, since scanners usually stand on level ground the angle between scanner and surface declines rapidly with increasing φ . The same is true for long walls, where the angle is a function of θ . Looking up points in shadow map pixels under a very small angle introduces a considerable sampling error, as illustrated in Figure 4a. This could be handled by increasing the threshold ε . However, a higher threshold also increases the rate of false negatives and is therefore to be avoided.

To overcome this problem, we reconstruct the surface equation during texture lookup by fitting a plane to the 7×7 -neighborhood of the respective shadow map texel. The distance from a point to this plane then gives a much better estimate for the visibility of said point (see Figure 4b).

For a plane given by a normal \mathbf{n} and a distance l from the origin in the local coordinate frame of a scanner and

Data:
attribute:

- Euclidean point coordinates \mathbf{p}_e ;

uniform:

- an array of n shadow maps S_i ;
- the positions \mathbf{s} of the n scanners

Result:

- a boolean flag c that indicates, whether the point is an artifact;

$c = \text{false};$

forall the shadow maps S_i **do**

 calculate $(\hat{\theta}, \hat{\phi})$ from \mathbf{p}_e ;

if $\text{distance}(\mathbf{p}_e, \mathbf{s}_i) + \varepsilon < S_i(\hat{\theta}, \hat{\phi})$ **then**
 | $c = \text{true};$

end

end

Algorithm 1: Shader pseudocode for naive artifact removal. The ε is a tolerance threshold that has to be defined with respect to the data range. The result of this approach can be seen in Figure 6a

a point \mathbf{p} in the same coordinate frame we can compute the deviation d of the point as follows:

$$d = \langle \mathbf{n}, \mathbf{p} \rangle - l \quad (1)$$

This solves the problems we experienced in areas with a large but constant depth gradient, for example roads or walls.

3.3 Missing Data

Although we used a smaller size for the shadow map than the original spherical scanner resolution it can still happen that pixel-sized holes remain in the shadow map. Since we are approximating planes over the neighborhood of each pixel this does not pose a serious problem. To make sure that the map is mostly filled, we use a point size of two pixels during rendering of the shadow maps.

The larger problem is how to cope with missing data in the texture. There are three principal reasons for not having any range data from the scanner for a given (θ, φ) :

1. The laser beam did not hit any geometry within the maximum distance for the scanner. Consequently, the beam is classified as background.
2. The beam did hit reflecting or scattering geometry, seen at the bottom right of Figure 3, where the note-

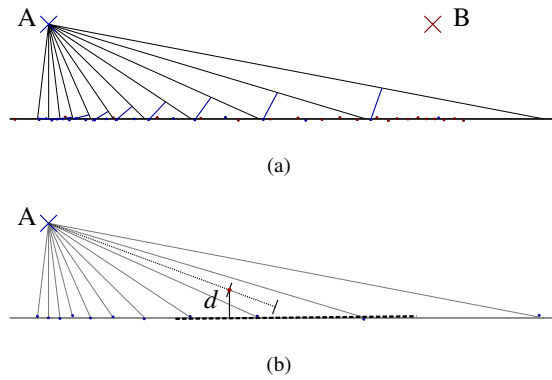


Figure 4: Illustration of the anisotropy problem in spherical shadow maps. (a) The distance from scanner A in the region of densest sampling of scanner B does not suffice to make a decision for the points of B without increasing the threshold considerably. (b) Fitting planes to the neighborhood allows the computation of the exact divergence for a given point p . Here the blue points have been reconstructed from the shadow map information and a plane (dashed) was fitted to the neighborhood of the pixel to which p was projected. Calculating the distance d to this plane gives a much better estimate of the actual distance than the average distance to the shadow map values (dotted line), especially under acute angles.

book screen and the reflective bands of the cone are missing.

3. The data has already been processed in some way, which was the case with our dataset, where some – but by no means most – erroneous geometry has been removed by hand.

If it was only for the background being hit, we could treat such shadow map pixels as infinitely far away and

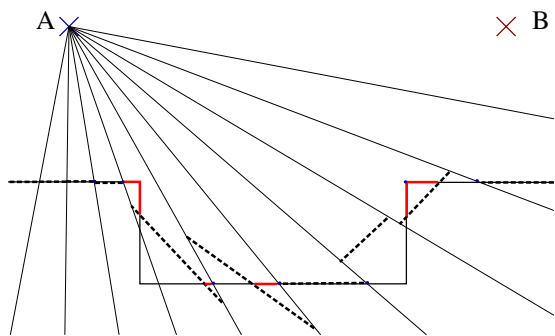
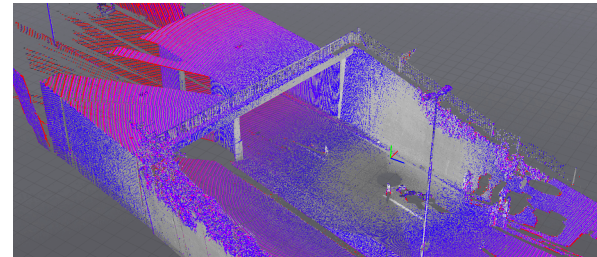
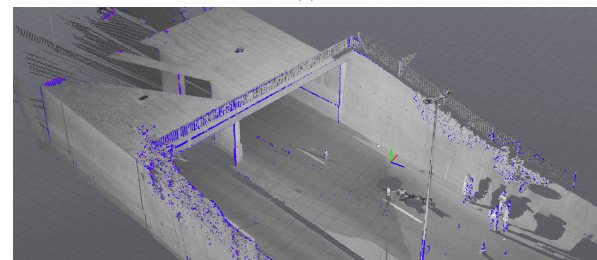


Figure 5: A 2D example for the sensibility of fitting planes in the vicinity of edges. For every texel of the shadow map of scanner A the best fitting plane to its 3-neighborhood is shown as a dashed line, indicated in red are the areas that would have a positive distance to their respective planes, putting them at risk to be classified as outlier.

discard any point in the final dataset that would correspond to this pixel (since obviously at least one scanner saw the background through this point). However, since there are other possible causes for missing data we chose to disregard such empty areas and to make no assumptions on the visibility of any point that would be mapped to there.



(a)



(b)



(c)



(d)

Figure 6: The different problems we encountered and the solutions found by our algorithm: a) Actual distance minus map distance for a scan tested against its own shadow map using Algorithm 1 and the same when using fitting planes (b). Note that there are still misclassification at the corners, even though they are very small. c) After applying the confidence map nearly all misclassifications have been eliminated. d) The introduction of a second scan facilitates recognition of first artifacts (color coded from blue to red is the respective confidence value).

3.4 Confidence Map

One problem that remains, occurs in areas with diverging gradients, mainly edges and corners, where there are still issues caused by the shadow map resolution. Usually the shadow map resolution is considerably smaller than the resolution of the original dataset, which introduces large errors in the computation of the osculating planes, leading to misclassifications in the vicinity of corners and edges (see Figure 5).

To facilitate more control over the classification we abandon the binary classification of Algorithm 1 in favor of a real certainty value that indicates how likely a given point is an artifact, with $c = 1$ meaning that the point is an artifact for sure and $c = 0$ meaning that the point surely belongs to the model. We can then sum up the weighted distances over all shadow maps and apply a threshold to the mean c to infer a binary classification.

The actual weighting of the classification is done using an additional confidence map for each shadow map, which is 0 on edges between valid points and background and otherwise, in areas that are completely covered by the shadow map, is computed as $1 - \tilde{\sigma}$:

$$\tilde{\sigma}_{u,v} = \sqrt{\frac{1}{|N|} \sum_{i=1}^{|N|} (\langle \mathbf{n}, \mathbf{p}_i \rangle - l)^2};$$

with \mathbf{p}_i being a reconstructed point from the neighborhood N of a given texel at (u, v) and \mathbf{n}, l being the parameters of the fitted plane, as in Equation 1. This is nothing else than the root mean square error (RMSE) of the fitted plane for a texel. The final confidence value e for a texel is therefore:

$$e_{u,v} = 1 - \begin{cases} \tilde{\sigma}_{u,v} & \Leftrightarrow \text{valid} \\ 1 & \Leftrightarrow \text{else} \end{cases} \quad (2)$$

A shadow map texel is considered *valid*, if the four 3×3 -corners of the pixel contain enough points to be able to robustly fit a plane to them also (we only required the corners to contain at least four points each themselves, the planes are not actually reconstructed). Otherwise we can assume that there is a large patch of background in the neighborhood of the given texel and filter these border areas out.

Since it can be computed together with the planes themselves, the confidence map does not have to be stored explicitly. Here it is only given for convenient illustration (see Figure 7).

After these optimizations we arrive at the refined Algorithm 2.

Overall the algorithm needs none of the usual preprocessing steps like constructing a space partitioning or estimating normals. It simply works by streaming all available data through the GPU twice, comparing the

Result:

- a confidence value c that indicates, how likely the point is an artifact;

$c = 0;$

$j = 0;$

forall the shadow maps S_i **do**

 calculate $(\hat{\theta}, \hat{\phi})$ from \mathbf{p}_e ;

 reconstruct set of Cartesian coordinates N over the neighborhood of $S_i(\hat{\theta}, \hat{\phi})$;

 fit a plane P to the points in N ;

 calculate d using (1);

 calculate e using (2);

if $d > 0$ **then**

$c = c + (e * d);$

$j++;$

end

end

$c = \frac{c}{j};$

Algorithm 2: Shader pseudocode for the refined classification algorithm. The input is the same as in Algorithm 1.

points with each shadow map in the second run, which yields a total runtime in the order of $O(k \cdot n)$, with $k \ll n$ being the number of scans. However, the shadow maps lie in fast GPU memory and the lookups therein are done using specialized hardware units, so the dominant factor is clearly n . The following section gives detailed timings and classification results for a real world dataset with 138 million points.

4 RESULTS AND DISCUSSION

We tested our approach on a notebook using an Intel i7 Quad-core processor with 2,3 GHz, and an Nvidia GeForce GTX 485m graphics card, running a 64bit Linux. We deliberately chose mobile hardware since we wanted the software to be used under field conditions. The data stems from an architectural scan of a bridge and consists of 138 million points distributed over five scans.

The timings we achieved (see Table 1) were very satisfying. In fact, the actual processing time carries little weight compared to the I/O latency of the hard drive. Although this could be alleviated to some extent by using for example an SSD it would still dominate the running time of the algorithm.

Classification quality depends highly on certain design choices. We present a comparison of results with the different optimization steps of Section 3 in Figure 6. There the progression of the algorithm through Sections 3.1 to 3.4 is depicted on an example dataset (the same dataset that was used to generate the example shadow and confidence maps). One has to note that

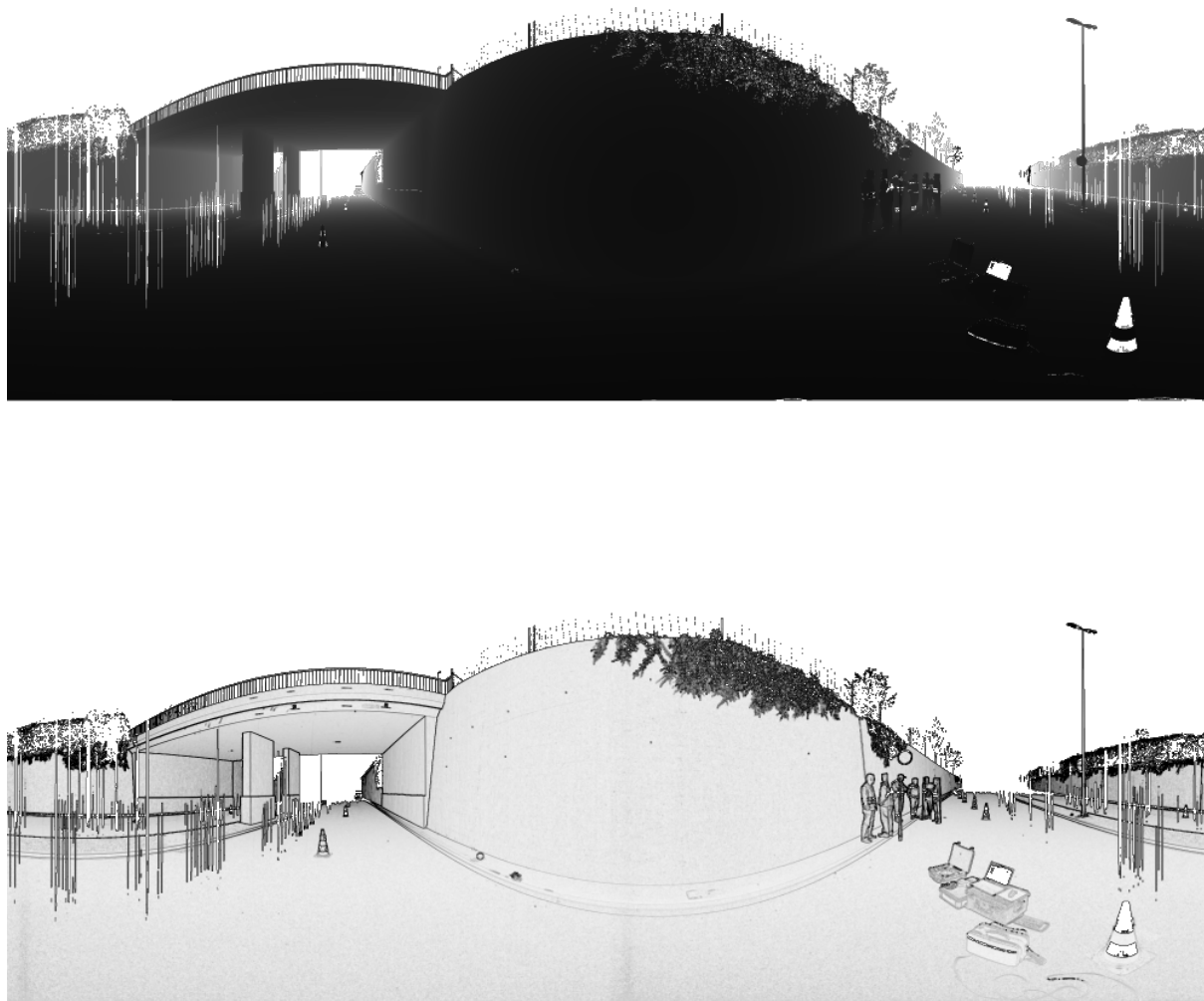


Figure 7: The spherical shadow map produced from the scan seen in Figure 3 (top) and the respective confidence map (bottom). One can see that values taken from texels that constitute edges are assigned a lower weight due to the confidence map and therefore do not compromise the artifact classification.

the confidence value introduced in Equation 2 was chosen for data on a centimeter-scale. That means that the RMSE for different scales has to be adjusted to this, since the deviations may have very small fractional values otherwise, making the confidence extremely high. In our case the RMSE can be interpreted as deviation from the plane in centimeters, implying that all planes with a RMSE higher than 1 cm are rejected (confidence 0).

Another issue that is obvious is that knowing the *exact* scanner position for each scanner is crucial to the performance of the algorithm. If this information is lost after registering the scans, one has to put some effort

into aligning the positions exactly. If a new scan is being produced, however, this information should be easily accessible and can be exported into the data. An additional prerequisite is an as-exact-as-possible registration of the data. Otherwise large portions of the dataset may receive a positive c that has to be thresholded appropriately (see Figures 9a). Normally the relative scanning position for each scan is in the origin, making the generation of the maps easy. The absolute positions in the scene are then found during registration. This implies that in order to account for a maximum misregistration of x cm the minimum threshold has to be at least x cm also.

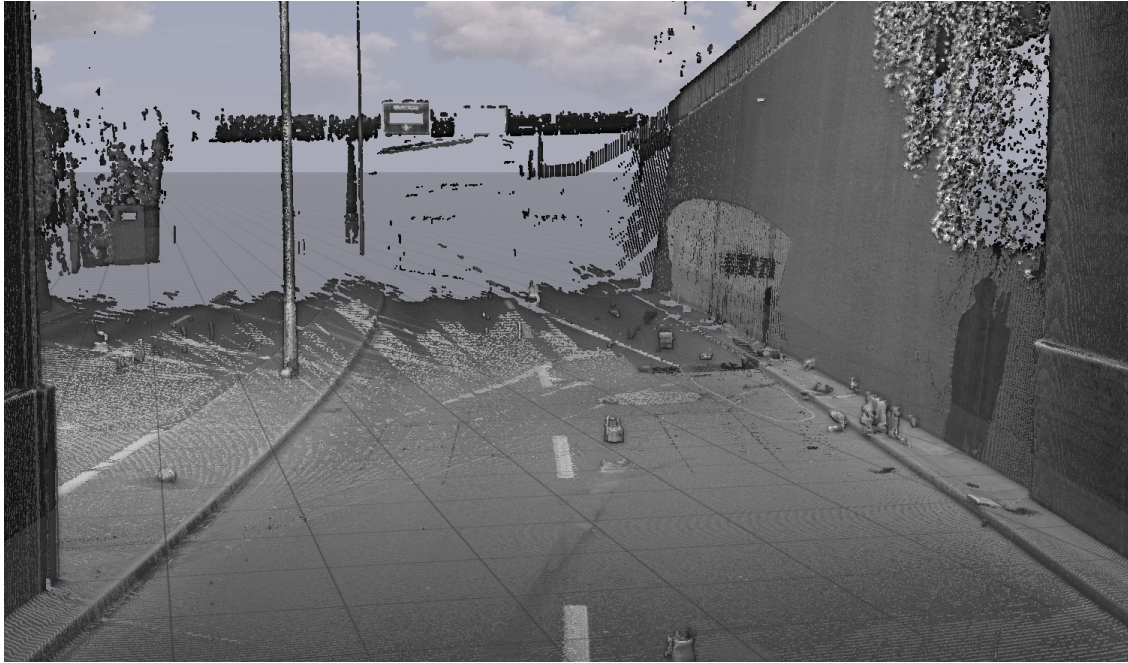


Figure 8: The same view as Figure 1 after applying our artifact removal procedure. The people and the car have been completely removed. However they may leave a shadow, if the lighting in two merged scans differed considerably. Additionally, nearly all of the stripes and parts of the cones were removed.

Scan	# Points	Load	Init	Classify
0	23.0m	5 699 ms	48 ms	1 852 ms
1	38.5m	9 482 ms	82 ms	3 265 ms
2	23.5m	6 414 ms	49 ms	1 943 ms
3	38.2m	9 279 ms	82 ms	3 099 ms
4	15.2m	3 766 ms	31 ms	1 318 ms
Σ	138.4m	34 659	292	11 477
Total Time		80 s ($\approx 2 \cdot \text{Load} + \text{Init} + \text{Classify}$)		

Table 1: Processing times for a 138 million point dataset. The complete processing took approximately 80 seconds. In the table the times are divided in "Load", i.e. the transfer of the data from the hard drive to the GPU, "Init", i.e. the initialization and rendering of the shadow-map and "Classify", i.e. the classification of all points of the dataset according to the shadow-maps of all scans using Algorithm 2. The classification has to use all shadow maps, but since they are of equal size the classification time per map equals to the total time divided by the number of scans, i.e. $\frac{\text{classification time}}{5}$ in our case. Note that in the total processing time the "Load" component appears twice, since we streamed the data from the hard drive again for classification.

A peculiar observation we made was that some of the objects in the scene, in particular a traffic cone under the bridge, have only been slightly moved in between scans – probably by accident. This slight dislocation allowed for a removal of one of the partial cones (Figure 9b). Knowing this, it might be beneficial to at least

slightly move equipment that can for some reason not be removed completely from the scene in between the scans.

5 CONCLUSION

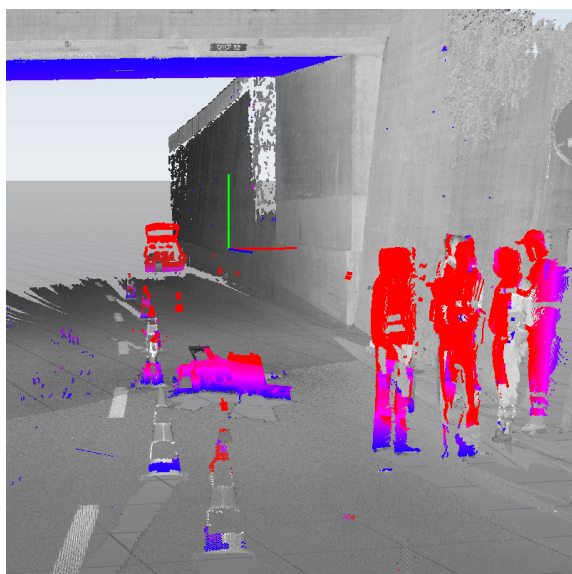
We have presented an approach for artifact classification in large point clouds comprised of multiple scans. The algorithm can for the most part be implemented on the GPU with linear asymptotic complexity. It results in a confidence value for each point that can easily be evaluated by the user using color coding. With this information the user is able to choose a threshold via a slider to remove the found points. Thanks to the edge-sensitivity it works very conservative, although that means that certain artifacts can remain in the dataset, because no other scanner could confidently reject them. However, since the classification can be done within minutes, this can also be used to infer a position for additional scans during a campaign. To improve the robustness of the approach, slight dislocations of light equipment between scans can already have a huge effect.

6 ACKNOWLEDGMENTS

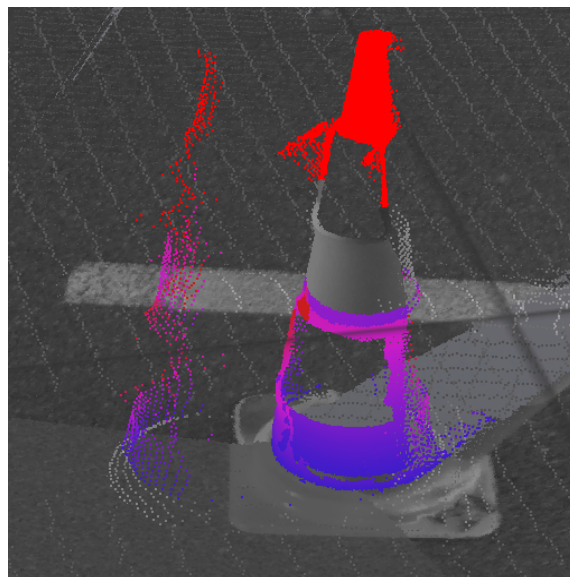
The authors would like to thank the enerotec engineering AG (Winterthur, Switzerland) for providing us with the data and for their close collaboration. This work was partially funded by EUREKA Eurostars (Project E!7001 "enercloud - Instantaneous Visual Inspection of High-resolution Engineering Construction Scans").

7 REFERENCES

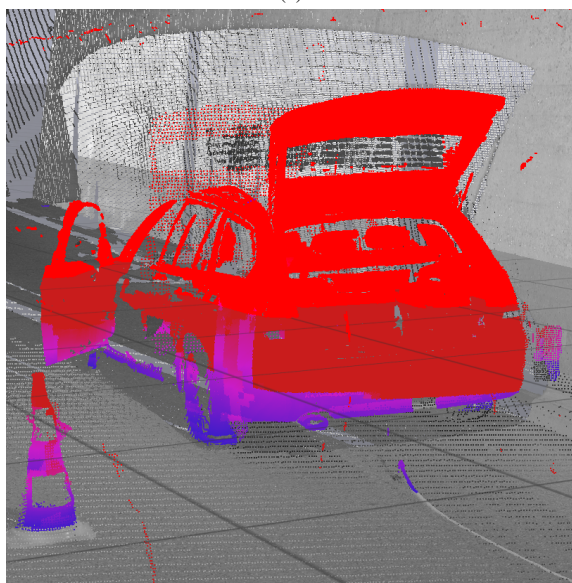
- [ARL⁺10] O. Alexander, M. Rogers, W. Lambeth, J. Chiang, W. Ma, C. Wang, and P. Debevec. The Digital Emily Project: Achieving a Photorealistic Digital Actor. *IEEE Comput. Graph. Appl.*, 30(4):20–31, July 2010.
- [BGM⁺09] F. Bettio, E. Gobbetti, F. Marton, A. Tinti, Emilio Merella, and Roberto Combet. A Point-based System for Local and Remote Exploration of Dense 3D Scanned Models. In Debattista et al. [DPPS09], pages 25–32.
- [CM92] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145 – 155, 1992.
- [DPPS09] Kurt Debattista, Cinzia Perlingieri, Denis Pitzalis, and Sandro Spina, editors. *VAST09: The 10th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, St. Julians, Malta, 2009. Eurographics Association.
- [DRL10] P. Dobrev, P. Rosenthal, and L. Linsen. Interactive Image-space Point Cloud Rendering with Transparency and Shadows. In Vaclav Skala editor, *Communication Papers Proceedings of WSCG, The 18th International Conference on Computer Graphics, Visualization and Computer Vision*, pages 101–108, Plzen, Czech Republic, 2 2010. UNION Agency–Science Press.
- [GPAR04] Markus Gross, Hanspeter Pfister, Marc Alexa, and Szymon Rusinkiewicz, editors. *SPBG'04 Symposium on Point - Based Graphics*, Zürich, Switzerland, 2004. Eurographics Association.
- [GSS08] L. Grosman, O. Smikt, and U. Smilansky. On the application of 3-D scanning technology for the documentation and typology of lithic artifacts. *Journal of Archaeological Science*, 35(12):3101–3110, 2008.
- [IM09] L. Iuliano and P. Minetola. Enhancing moulds manufacturing by means of reverse engineering. *The International Journal of Advanced Manufacturing Technology*, 43(5–6):551–562, 2009.
- [KLR12] T. Kanzok, L. Linsen, and P. Rosenthal. On-the-fly Luminance Correction for Rendering of Inconsistently Lit Point Clouds. *Journal of WSCG*, 20(2):161 – 169, 2012.
- [KNRS12] J. Köhler, T. Nöll, G. Reis, and D. Stricker. Robust Outlier Removal from Point Clouds Acquired with Structured Light. In *Eurographics 2012-Short Papers*, pages 21–24. The Eurographics Association, 2012.
- [LNCV10] J. L. Lerma, S. Navarro, M. Cabrelles, and V. Villaverde. Terrestrial laser scanning and close range photogrammetry for 3D archaeological documentation: the upper palaeolithic cave of parpalló as a case study. *Journal of Archaeological Science*, 37(3):499–507, March 2010.
- [MT09] H. Masuda and I. Tanaka. Extraction of Surface Primitives from Noisy Large-Scale Point-Clouds. *Computer-Aided Design and Applications*, 6(3):387–398, 2009.
- [PKGFO3] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 315–326, Los Alamitos, CA, USA, 2003. IEEE Computer Society.
- [PMG04] M. Pauly, N. J. Mitra, and L. J. Guibas. Uncertainty and Variability in Point Cloud Surface Data. In Gross et al. [GPAR04], pages 77–84.
- [PV09] S. Pu and G. Vosselman. Knowledge based reconstruction of building models from terrestrial laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(6):575–584, November 2009.
- [RHHL02] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 438–446, New York, NY, USA, 2002. ACM.
- [SBS05] O. Schall, A. Belyaev, and H. Seidel. Robust filtering of noisy scattered point data. In *Proceedings of the Second Eurographics / IEEE VGTC conference on Point-Based Graphics*, SPBG'05, pages 71–77, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.
- [Sot06] S. Sotoodeh. Outlier detection in laser scanner point clouds. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVI-5*, pages 297–302, 2006.
- [SZW09] C. Scheiblauer, N. Zimmermann, and M. Wimmer. Interactive Domitilla Catacomb Exploration. In Debattista et al. [DPPS09], pages 65–72.
- [WBB⁺08] M. Wand, A. Berner, M. Bokeloh, P. Jenke, A. Fleck, M. Hoffmann, B. Maier, D. Staneker, A. Schilling, and H. Seidel. Processing and interactive editing of huge point clouds from 3D scanners. *Computers & Graphics*, 32(2):204–220, 2008.
- [WPK⁺04] T. Weyrich, M. Pauly, R. Keiser, S. Heinzele, S. Scandella, and M. Gross. Post-processing of Scanned 3D Surface Data. In Gross et al. [GPAR04], pages 85–94.



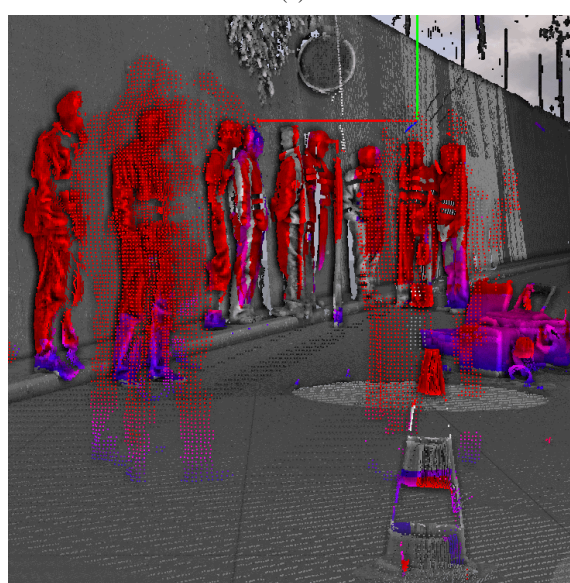
(a)



(b)



(c)



(d)

Figure 9: Some classifications inferred by our algorithm. For these images a threshold of 2 cm was used. The blue area on the ceiling in (a) is due to inaccurate registration of the scans and has to be taken care of by choosing an appropriate threshold. The cone in (b) was slightly displaced during scans, allowing at least the artifact from one scan (left) to be completely recognized. c) The car was apparently completely removed for at least one scan, which made it almost completely recognizable. Note the slight shadow of the tailgate, indicating that this car was slightly displaced also present in a second scan. d) A group of people that was present at roughly the same place during different scans. Note that parts of the group in the background could not be classified, since they were shadowed by other people closer to the scanners (Shading was added to improve geometric structure perception).

Identification of Abnormal Cervical Regions from Colposcopy Image Sequences

Mingpei Liang¹, Gaopin Zheng², Xinyu Huang¹, Gaolin Milledge¹, Alade Tokuta¹

¹North Carolina Central University ²Shenzhen Luohu Maternity and Infant Healthy Institute

1801 Fayetteville Street

2013 Taibai Rd, Luohu

Durham, NC 27707, USA

Shenzhen, Guangdong 518999, P.R. China

{mliang, huangx, gzheng, atokuta}@ncu.edu

gaopinzheng@yahoo.com.cn

ABSTRACT

Cervical cancer is the third most common cancer in women worldwide and the leading cause of cancer death in women of the developing countries. Cancer death rate can be greatly reduced by regular screening. One of the steps during a screening program is the detection of the abnormal cells that could evolve into cancer. In this paper, we propose an algorithm that automatically identifies the abnormal cervical regions from colposcopy image sequence. Firstly, based on the segmentation of three different image regions, a set of low-level features is extracted to model the temporal changes in the cervix before and after applying acetic acid. Second, a support vector machine (SVM) classifier is trained and used to make predictions on new input feature vectors. As the low-level features are very insensitive to accurate image registration, only a rough normalization step is needed to sample image patches. Our preliminary results show that our algorithm is accurate and effective. Furthermore, our algorithm only needs to sample patches from six image frames within a five-minute time period. Hence, the proposed algorithm also could be applied to improve the accuracy of the mobile telemedicine for cervical cancer screening in low-resource settings.

Keywords

Colposcopy Image Processing, Support Vector Machine, Feature Extraction, Cervical Cancer

1. INTRODUCTION

Cervical cancer is the commonest cause of cancer death among women in developing countries [QBP+12]. Cervical cancer is preceded by pre-malignant cervical intraepithelial neoplasia (CIN). In order to prevent cervical cancer, the accurate diagnosis of cancer cells and abnormal regions followed by appropriate therapy is necessary. Colposcopy is a widely used diagnostic method to detect CIN and cervical cancer. Once the abnormal region is identified, the samples of cells are often taken from the cervix for an abnormal cytological screen (i.e., Papanicolaou smear). During the colposcopic exam, application of 3-5% acetic acid to

the cervix can turn abnormal and metaplastic epithelia to white, while normal cervical squamous epithelia remain pink. It is believed that the amount of whiteness is positively associated with the severity of cervical intra-epithelial neoplasia (CIN) [GKL+11, QGR+11]. Therefore, the whiteness is considered as one of major characteristics to detect cancer and pre-cancerous regions. Other features, such as morphologic characteristics around precursor lesions [Lan05] and vascular patterns, can also be used for identification. However, they may not be salient sometimes and thus may not be as effective as whiteness of cervical epithelia.

In order to minimize subjective variability among physicians and improve reliability and repeatability of diagnosis, a sound computer aided image processing algorithm that could combine all possible features is highly desirable. Such a system can enhance the power of existing colposcopes and would make them immediately useful in low-resource areas of the developing countries [SCN+09].

In this paper, we propose an image processing and statistical learning procedure to identify abnormal

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

regions in colposcopy images. Firstly, we notice that the cervical surface has limited and very similar texture patterns. Some regions could also be very smooth. Thus, many image registration techniques that rely on detection of feature points could fail since feature points often cannot be detected and tracked over image frames consistently. Our algorithm is particularly useful for the situation when the accurate image registration cannot be achieved. Second, a set of low-level features is extracted to model color and thickness changes in the cervix. The Support Vector Machine [CV95] is applied for the classification. Based on our preliminary experiments, accurate classification rate can be achieved by sampling only six images within a five-minute time interval. We believe that the proposed algorithm could be adopted to improve the diagnostic performance of the mobile telemedicine for cervical cancer screening [QGR+11]. As number of current training data is limited and more data is often needed to improve robustness of classification performance, we also plan to label and train on a large set of image sequences in the future.

The remainder of this paper is organized as follows. Section 2 describes related work. Our proposed algorithm is given in section 3. Experiments in section 4 show the success classification rates. The conclusion is given in section 5.

2. RELATED WORKS

The research in this area is limited and current methodologies [AK07, AKL11, SCN+09] follow the standard pipeline of medical image processing.

The first step is the image registration. In [AK07], feature points are detected using the Harris corner detector. This detector can detect changes in the first derivative of an image. Other similar feature points detector, such as SIFT [Low04] and MSER [FL07], also could be applied with similar detection performance. Matched feature points in two consecutive image frames are used to estimate a 2D transformation between them. However, many colposcopy images contain limited and very similar texture patterns, thus, matching of feature points

often are not robust and accurate.

The registration algorithm based on a rigid transformation including translation and in-plane rotation is described in [AKL11]. Since a larger image region is used for registration, the performance could be better than the algorithms based on detection of feature points. However, during the colposcopic exam, other rigid transformations of the camera are also very common, such as scaling and out-of-plane rotation. Moreover, the cervix itself could have nonlinear deformations. Hence, only translation and in-plane rotation are unable to model the transformations in all the image frames. If all the transformations are considered, the process could be time-consuming and often done offline. The speed of the registration algorithm in [AKL11] is approximately 10 minutes for two image frames.

In the feature extraction step, whiteness is measured based on changes of saturation values in [AKL11]. Other features, such as morphologic features and vascular patterns, are not modeled and measured in most existing algorithms.

3. OUR ALGORITHM

According to [GZH04], the cervix area could be divided into three regions, squamous epithelium that remains pink after acetic acid application, columnar epithelium that is a dark and irregular region between endometrium and squamous epithelium, and the acetowhite region that is the region turned into white after acetic acid application. Figure 1 shows the structure of cervix area. The region of specular reflection is also large and keeps changing its position due to movements of colposcopy and cervix.

Our algorithm can be divided into five modules: 1) Specular reflection removal; 2) Segmentation of different image regions; 3) Normalization; 4) Extraction of low level features; 5) Classification using Support Vector Machine. The normalization step is mainly used to sample image patches for the SVM prediction. For the training purpose, this step could be skipped. Figure 2 gives an overview of the

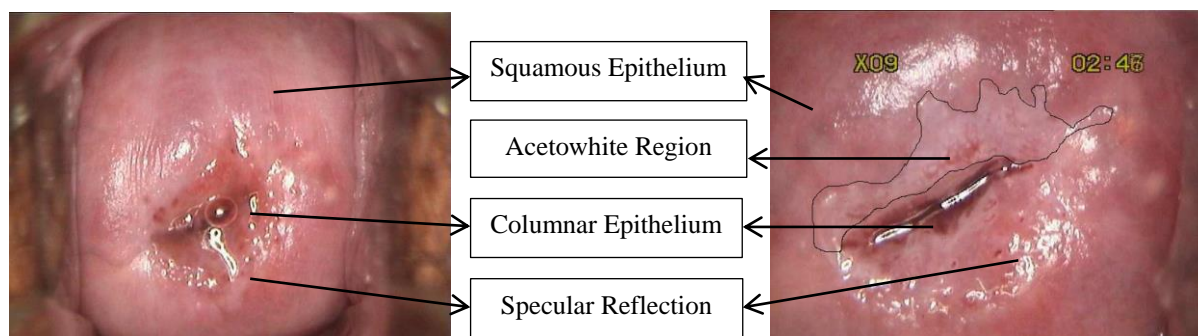


Figure 1: Image before acetic acid application (Left). Image enlarged 9 times and taken 2 minutes 46 seconds after acetic acid application (Right).

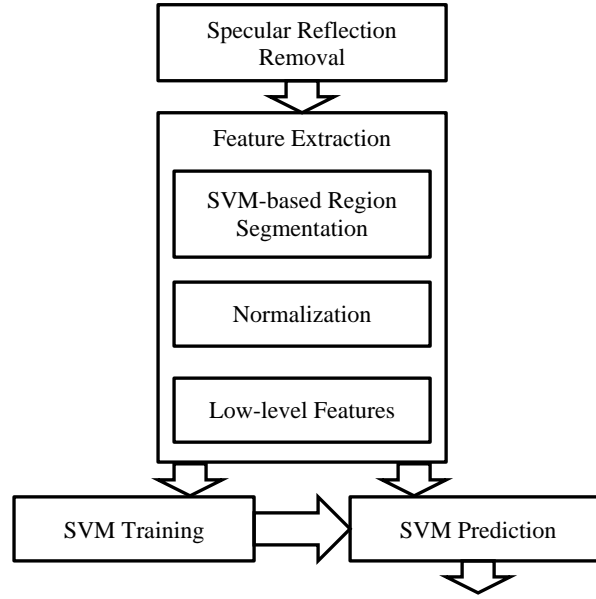


Figure 2: Outline of the algorithm.

algorithm.

3.1 Specular Reflection Removal

Specular reflections are random patterns that would greatly affect the estimation of low level features. Therefore, it is important to remove them before any further processing. Specular reflections are the brightest pixels in the image and the region containing specular reflections usually is smooth. We use a bi-linear interpolation method to fill the reflections. First, we compute a binary reflection map by a threshold (i.e., 250 in our algorithm). For each reflection pixel (x_0, y_0) , we then search four envelop points that are outside the region of reflections, I_l , I_r , I_t , I_b , along vertical and horizontal directions. The reflection pixel is filled by

$$I_{(x_0, y_0)} = \frac{I_l \cdot (x_r - x_0) + I_r \cdot (x_0 - x_l) + I_b \cdot (y_0 - y_t) + I_t \cdot (y_b - y_0)}{2(x_r - x_l) + 2(y_b - y_t)}$$

where (x_r, y_0) , (x_l, y_0) , (x_0, y_b) , (x_0, y_t) are the pixel coordinates of four envelop points.

3.2 Feature Extraction

Given an image patch, since the accurate registration is not easy to obtain, we employ the use of low-level features that are not sensitive to the accuracy of registration. As mentioned in section 1, there are mainly three characteristics used to determine whether the patch contain abnormal cells and cancer

cells: whiteness changes of metaplastic epithelia, morphologic changes around precursor lesions, and changes of vascular regions. To model these characteristics, we propose six low-level features based on color, edge information, and texture information.

3.2.1 SVM-based Region Segmentation

In order to measure the color changes correctly, we need to separate the regions with bleeding tissues that often become worse after acetic acid application. Otherwise, these bleeding regions could skew the actual whiteness changes. We sampled dozens of pixels in bleeding regions, the dark region of columnar epithelium, and other regions. Figure 3 shows the color distributions of three regions.

Firstly, we build a classifier to separate bleeding regions and the dark region of columnar epithelium. Given training set of colors $x_i \in R^3$ ($i = 1, \dots, N$) and corresponding labels $y_i \in \{-1, 1\}$ (i.e., bleeding region and other two regions), the SVM training algorithm is used to find the best decision plane that separates the largest subset of the training colors correctly and maximizes the margin between sampled colors. A new color sample x is classified by the decision plane, given by

$$y(x) = \sum_{i=1}^N \alpha_i y_i k(x, x_i) + b$$

where α_i denotes the Lagrange multipliers, b is a bias term, N is the number of training colors, and $k(x, x_i)$ is the kernel function. α_i and b are estimated from the SVM training stage and a linear kernel function is used.

Similarly, another SVM classifier is built to separate the dark region of columnar epithelium and the third region.

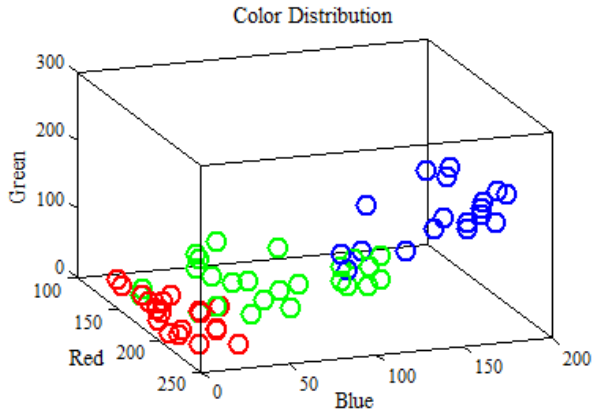


Figure 3: Color distributions of three different regions. (Red) samples from bleeding regions. (Green) samples from dark region of columnar epithelium. (Blue) samples from other regions.

3.2.2 Image Normalization for Prediction

In order to obtain six temporal image patches for the SVM predication, which contain roughly same cervical area, a normalization step is used. Unlike the image registration, this normalization step only gives a quite rough alignment among six image frames.

First, we use the SVM classifier learned in previous section to estimate the dark regions of columnar epithelium. The largest connected component in the central area of the image is considered as the dark region of columnar epithelium. The centroid of the connected component is used to compute the translation among image frames. The 1st principal component from principal component analysis (PCA) is used to compute the rotation and scaling parameters. Therefore, these image frames are roughly aligned together and we can easily extract feature vectors from corresponding image patches for predictions.

3.2.3 Low-level Features

After the separation between bleeding regions and other two regions, 1st quartile, median, and 3rd quartile of pixel colors are computed in both regions respectively. These two 3×1 vectors are used to represent colors of two regions in the image patch of each image frame.

The edges are a strong clue for the morphologic changes around precursor lesions. A Canny edge detector [Can86] is applied to the image patch and the edge length, which is the number of edge pixels in the image patch, is computed.

Texture feature could be used to measure thickness of boundaries of precursor lesions since these boundaries often become thick and obvious after acetic acid application. We compute the mean value of gradient magnitudes in the image patch, $\frac{1}{N} \sum_{x_i, y_i} \sqrt{J_{x_i}^2 + J_{y_i}^2}$, where N is the total number of pixels in the patch and (J_{x_i}, J_{y_i}) is the gradient for the i th pixel. If the cervix surface is smooth, then the mean magnitude is small in general.

Figure 4 shows the temporal distributions of median color in the non-bleeding region, edge length, and the texture feature. These distributions are the average distributions of 20 sampled patches for each image frame. From these distributions, we can find that median of green channels of abnormal regions tends to increase after acetic acid application. The distribution of median of blue channels is also very similar to Figure 4(a). The medians of red channels tend to remain same for both regions as shown in Figure 4(b). As a result, the color of the abnormal region tends to turn into white. We also notice that edge length and texture features also increase in the abnormal regions as shown in Figure 4(c) and 4(d), which indicates the morphologic changes. Since the distribution for a single patch could vary, it is necessary to concatenate all the features together to form a more robust feature vector. Thus, there is an 8×1 vector for each image patch and 6 image patches; and thus, the size of the feature vector is 48×1 .

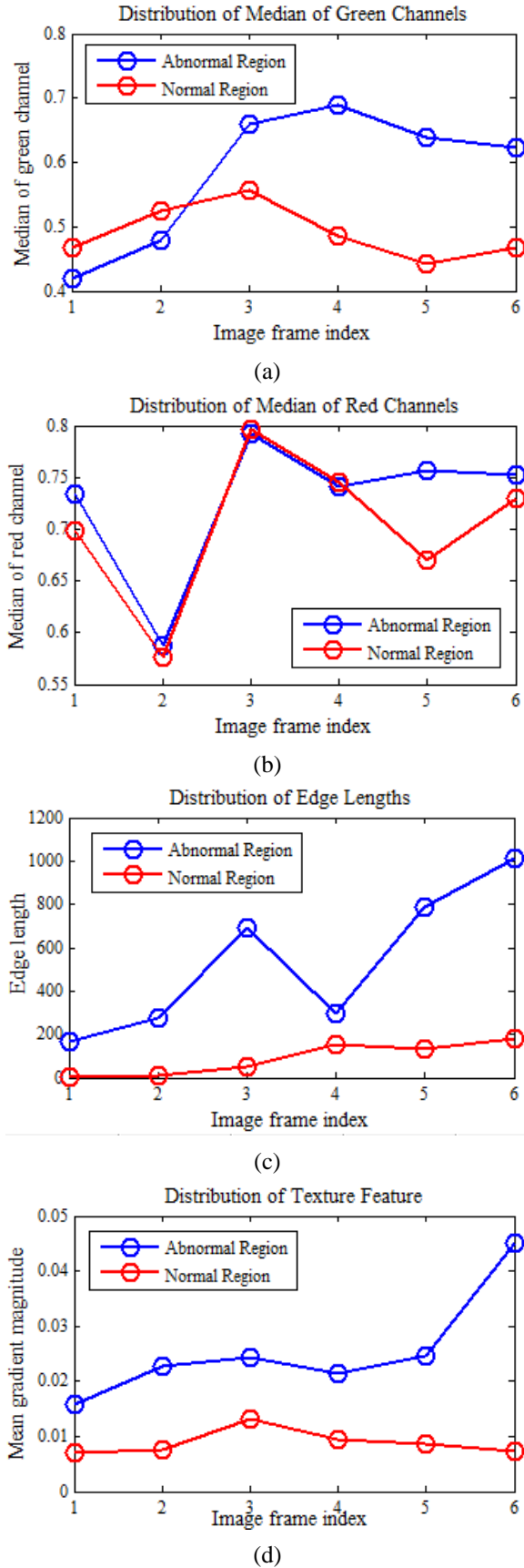


Figure 4: Temporal distributions of different features.

3.3 SVM Classification

In the training stage, we sample a set of image patches that contain abnormal regions and normal regions. The image patch size is consistent with the actual size for the Papanicolaou smear. Due to the irregular shape of the abnormal region and scaling factor of each image frame, the resolution of the image patches varies. We manually select and crop the corresponding image patches in the six image frames. Every set of six image patches contains the same cervix tissues. As the accurate image registration is not required, these patches often have different image appearances with unknown in-plane and out-of-plane rotations, translation, and scaling. However, this would not affect the classification performance since our low-level features are not sensitive to these linear or nonlinear transformations.

Given $N \times M$ training data $x_i \in R^M$ ($i = 1, \dots, N, M = 48$) and observations $y_i \in \{-1, 1\}$ (i.e., abnormal region and normal region), the SVM algorithm estimates the hyperplane to separate two different results. Three different kernel functions, linear kernel, radial basis function (RBF) kernel, and polynomial kernel, are evaluated.

4. EXPERIEMNTS

4.1 Datasets

We manually cropped 48 sets of positive rectangular patches from cervical lesion region from 12 patients with various degrees of cervical dysplasia. We also cropped approximately 40 sets of negative patches from normal cervical regions from these patients. Each set of positive or negative patches contains patches cropped from six temporal image frames. These data sets are used for the SVM training stage. The patch resolution ranges from 67×67 to 118×168 , which is caused by different zooming factor of the colposcopy and different distances between cervix and the colposcopy. The patch size is also selected so that the cells in this region can be sampled for the subsequent Papanicolaou smear. Figure 5 and Figure 6 show two sets of image patches. We can find that the patch resolutions are not exactly same. However, they cover roughly the same cervical region. As the low-level features are not sensitive to the accurate image registration, our classification performance is not affected.



Figure 5: Example of one set of image patches with abnormal cells. They are arranged from left to right and from top to bottom in ascending order of time.

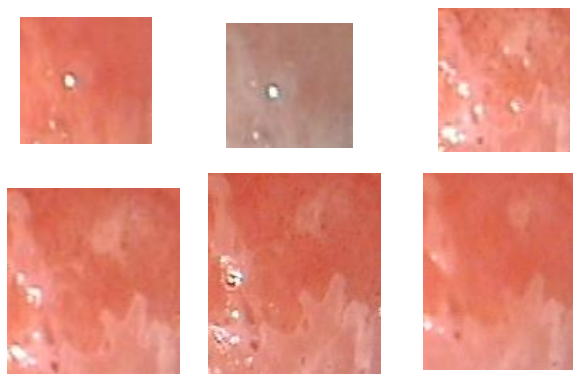


Figure 6: Example of one set of image patches with normal cells. They are arranged from left to right and from top to bottom in ascending order of time.

To evaluate prediction performance, we further sampled 30 sets of positive patches and 27 sets of negative patches.

4.2 Classification Performance

Table 1, 2, and 3 show the success classification rates when color features, edge features, and texture features are used separately. The performances are quite similar for three different kernels. Table 4 shows the success classification rate when all the features are combined. The highest rate from the testing dataset is 94.6% when the linear kernel is used. These experiments demonstrate the effectiveness of our proposed algorithm. Moreover, the simple linear kernel gives the best performance.

Table 1: Classification performances when only the features that model color changes are used.

Kernel	Success Classification Rate
Linear	83.9%
RBF	85.7%
Polynomial	85.7%

Table 2: Classification performances when only the edge features are used.

Kernel	Success Classification Rate
Linear	81.9%
RBF	85.1%
Polynomial	71.4%

Table 3: Classification performances when only the texture features are used.

Kernel	Success Classification Rate
Linear	87.2%
RBF	89.3%
Polynomial	83.4%

Table 4: Classification performances all the features are used.

Kernel	Success Classification Rate
Linear	94.6%
RBF	85.7%
Polynomial	89.3%

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a classification algorithm that can detect the abnormal cells in the cervical region. Our algorithm requires no accurate image registration. The low-level features are extracted and used as input in the SVM classification. The experiments show that the success classification rate could reach 94.6%. In the future, we would like to continue labeling data and further evaluate our algorithm using a large scale database. Since our algorithm is efficient and can run in real time, it would be promising to implement a real-time interactive diagnose system.

6. ACKNOWLEDGMENTS

The authors acknowledge support of the National Science Foundation HRD 0833184. The training and testing data sets are labeled by the physicians from Shenzhen Luohu Maternity and Infant Healthy Institute in Shenzhen, P. R. China.

7. REFERENCES

[AK07] Juan D. García Arteaga and Jan Kybic "Automatic landmark detection for cervical image registration validation", *Proc. SPIE 6514, Medical Imaging 2007: Computer-Aided Diagnosis*, 65142S (March 31, 2007)

[AKL11] J. D. Garcia A-Arteaga, J. Kybic, and W. Li, "Automatic colposcopy video tissue classification using higher order entropy-based image registration," *Comput. Biol. Med.*, vol. 41, pp. 960-970, 2011.

[Can86] J. Canny, "A computational approach to edge detection", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol: 8, Issue 6, pp 679-698, 1986

[CV95] Corinna Cortes and Vladimir Vapnik, "Support-vector networks", *Machine Learning*, Vol 20, Issue 3, pp 273-297, September 1995

[FL07] P.E. Forssen and D.G. Lowe, "Shape descriptors for maximally stable extremal regions", *IEEE International Conference on Computer Vision*, pp 1-8, 2007

[GZH04] S. Gordon, G. Zimmerman, and H. Greenspan, "Image segmentation of uterine cervix images for indexing in PACS", *Computer-Based Medical Systems, 2004, CBMS 2004. Proceedings, 17th IEEE Symposium on*, June 2004

[Lan05] H. Lange, "Automatic glare removal in reflectance imagery of the uterine cervix," in *Proc. SPIE 5747, Medical Imaging*, 2005, pp. 2183-2192.

[Low04] David G. Lowe, "Distinctive image features from scale-invariant keypoints", *International Journal of Computer Vision*, Vol 60, Issue 2, pp 91-110, November 2004

[QGR+11] Kelly E. Quinley, Rachel H. Gormley, Sarah J. Ratcliffe, Ting Shih, Zsolt Szep, Ann Steiner, Doreen Ramogola-Masire, and Carrie L. Kovarik, "Use of mobile telemedicine for cervical cancer screening", *Journal of Telemedicine and Telecare*, 2011

[QBP+12] M. K. Quinn, T. C. Bubi, M. C. Pierce, M. K. Kayembe, D. Ramogola-Masire, and R. Richards-Kortum, "High-Resolution Microendoscopy for the Detection of Cervical Neoplasia in Low-Resource Settings," *PLoS ONE*, vol. 7, p. e44924, 2012

[SCN+09] Y. Srinivasan, E. Corona, B. Nutter, S. Mitra, and S. Bhattacharya, "A Unified Model-Based Image Analysis Framework for Automated Detection of Precancerous Lesions in Digitized Uterine Cervix Images," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 3, pp. 101-111, 2009.

A facial motion tracking and transfer method based on a key point detection

Yasuhiro AKAGI
Kagoshima University
Korimoto, 1-21-24
Kagoshima
890-8580, JAPAN
akagi@ibe.kagoshima-u.ac.jp

Ryo FURUKAWA
Hiroshima City University
3-4-1, OzukaHigashi
AsaMinami-Ku, Hiroshima
731-3194, JAPAN
ryo-f@hiroshima-cu.ac.jp

Ryusuke SAGAWA
AIST
1-1-1 Higashi
Tsukuba, Ibaraki
305-8561 JAPAN
ryusuke.sagawa@aist.go.jp

Koichi OGAWARA
Wakayama University
Sakaedani 930
Wakayama-city
640-8510, JAPAN
ogawara@sys.wakayama-u.ac.jp

Hiroshi KAWASAKI
Kagoshima University
Korimoto 1-21-24
Kagoshima
890-8580, JAPAN.
kawasaki@ibe.kagoshima-u.ac.jp

ABSTRACT

Facial animation is one of the most important contents in 3D CG animations. By the development of scanning and tracking methods of a facial motion, a face model which consists of more than 100,000 points can be used for the animations. To edit the facial animations, key point based approaches such as "face rigging" are still useful ways. Even if a facial tracking method gives us all point-to-point correspondences, a detection method of a suitable set of key points is needed for content creators. Then, we propose a method to detect the key points which efficiently represent motions of a face. We optimize the key points for a Radial Basis Function (RBF) based 3D deformation technique. The RBF based deformation is a common technique to represent a movement of 3D objects in CG animations. Since the key point based approaches usually deform objects by interpolating movements of the key points, these approaches cause errors between the deformed shapes and the original ones. To minimize the errors, we propose a method which automatically inserts additional key points by detecting the area where the error is larger than the surrounding area. Finally, by utilizing the suitable set of key points, the proposed method creates a motion of a face which are transferred from another motion of a face.

Keywords

Facial Expression, Motion Transfer, Tracking, Non-rigid, Deformation

1 INTRODUCTION

A facial animation is one of the important topics in the area of computer vision and graphics[18, 15, 20, 21]. It is possible to obtain dense and accurate 3D points from an object with the development of 3D scanning method. In case of scanning a moving object, it is an important topic that how to detect a movement of a point from a frame to another frame. This kind of information is required for recognizing a facial expression and creating CG animations from the scanned point cloud.

Then, a number of methods have been proposed to detect the correspondences between frames. The techniques which track artificial markers on a target face are the well-known approaches to capture the motion of a face[3, 9]. In other cases, if it is difficult to place the artificial markers on a face, a method which detects a key point of a face can be a solution. There are two types of approaches to detect the key points: geometry based approaches and image based approaches. One of the solutions based on the geometry of a face is a non-rigid registration algorithm. The advantage of this approach is that it can deal with a deformable surface. If both a 3D shape and an image can be captured at the same time, the correspondence of points between frames can be found by using the image processing techniques. Moreover, by utilizing the key point of a face, methods called "facial transfer"[8, 11, 13, 7, 12] are proposed. These methods create a motion of an arbitrary face that reflects (copied from) motion of another face. To trans-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

for the facial motion, correspondences between a source face to a target one are required. Thus, the methods which can detect suitable pairs of key points between two faces are required.

2 RELATED WORK

Since it is difficult to create the facial animations manually, a number of methods are proposed to capture the facial motion. Two approaches have been researched to capture the facial motion: marker-based approaches and marker-less approaches. The marker-based approaches are more robust to detect motions than the marker-less approaches.

2.1 Marker-based approach of facial tracking

Huang *et al.* [9] proposed the method to capture a motion with high-fidelity by using one hundred markers. This method can capture dynamic wrinkles and fine-scale facial details. Bickel *et al.* [3] directly paint color point on a face to robustly detect the same points beyond frames. The marker-based approaches have a common problem that it is laborious to put the markers and it is difficult to capture the natural texture with the motion at the same time.

2.2 Marker-less approach of facial tracking

On the other hand, the marker-less approaches are proposed. Valgaerts *et al.* [18] proposed dense tracking method of movements of a face. This method uses a stereo-camera system and tracks the movement of each pixel of the camera images by using an optical-flow detection method. This method deforms a 3D face based on the optical-flow. Bradley *et al.* [4] proposed a facial tracking method which tracks both the movement of a texture and that of a geometry at the same time under the constant light condition. They also detect a shape of a mouth by using a facial parts recognition method. Sibbing *et al.* [15] uses the feature tracker like the KLT tracker to detect the motion of a face. Weise *et al.* [20] constructs the facial performance database of a person to detect the motion of the face from 2D image and 3D point set. Then, some approaches use the image processing methods to find the key points of a face such as a pupil of the eye, the outline of a lip, the tip of a nose and etc. The accuracy of these kind of methods is over 95% in some recent researches[5, 2].

2.3 3D shape based approach of facial tracking

The non-rigid registration algorithm is one of the useful methods to detect the motion of the point sets[17].

Jian *et al.* [10] proposed this kind of approach by using the L2 distance between Gaussian mixtures representing two point sets. However, since the features of the motion of a face vary in each facial part, it complicates the registration problem. If we can find a fine initial guess of the motion, the non-rigid registration algorithm will be a useful way to solve the problem.

2.4 Facial transfer

To create a motion of "an artificial face" such as animal characters, virtual humans and etc. in an animation, facial transfer(cloning) methods which copy the motion from a person to another virtual face are proposed[8, 11, 13, 7, 12, 14]. Huang *et al.* [8] utilizes a key point based deformation for a facial transfer. To represent motions of a face, a set of key points called Active Appearance Models(AAMs) are used. This method minimizes the deformations of AAMs to fit to a target face. Vieira *et al.* [19] proposed the facial transfer method which defines a zone of influence and a weight map for interpolating the movement of key points. Cosker *et al.* [6] generate a map for representing facial expressions (movements of the key points) based on Downhill Simplex Minimisation tracker. The map is calculate by analyzing training sets of the facial expressions with PCA. To perform the facial transfer, this method normalizes the movements of the key points and creates the weight vector between a source face and a target one. In this kind of the approaches, weighting values for key points are an important factor to represent the movement of a face. Moreover, since the targets of these methods are sparse polygon models or 2D images, a fitness of the shape in the area where there are no key points is not evaluated. This point can be a problem if these methods apply to dense face models.

3 OVERVIEW OF THE PROPOSED METHOD

We describe the overview of our study. The proposed method in this study consists of the following three steps.

- 1) **Initialization of key points tracking** In the paper, we assume the scanning method which can capture both 3D point clouds and 2D images at the same time. Therefore, we utilize the method proposed by Cao *et al.* [5] for finding facial landmarks from a 2D image of the face to detect key points in each frame as the initial tracking result for the following process.
- 2) **Non-rigid shape deformation for dense motion** In our method, a dense motion of a facial expression from an initial frame to other frames is represented by 3D non-rigid shape deformation. We utilize a Radial Basis Function based deformation

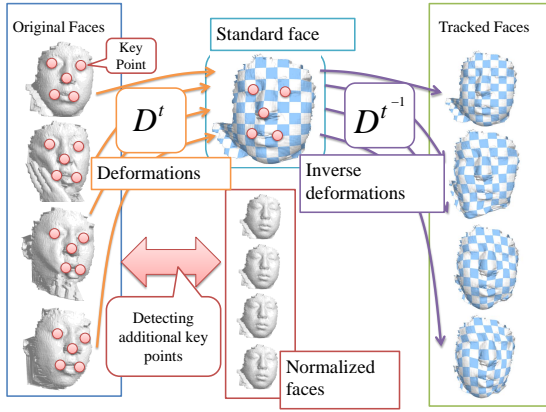


Figure 1: Overview of our approach for tracking a facial motion.

method [16]. The method generates deformed shapes from the shape of the initial frame to the other frames by using a pair of key points: the key points of the initial frame and those of the other frames (Figure1).

3) Extra key points addition Since the number of key points detected as the initial process is relatively small compared to all the 3D points, there is naturally a considerable amount of errors existing between scanned shapes and the deformed ones. To solve this problem, our method automatically detects additional key points to decrease the errors. Our method searches an area where errors between the scanned shapes and the deformed ones are larger than other areas to find the additional key points. By repeating the process, errors are minimized.

4 DEFORMATION METHOD OF A FACE

We describe the details of a deformation method of a facial motion. In the paper, F^t denote a vector of 3D points (p_0^t, \dots, p_n^t) of the face in the t -th frame. Then, our purpose of the facial motion tracking is to find deformations from F^0 to F^t . In the following section, we describe a basic idea of the deformation method.

4.1 Shape deformation based on the Radial Basis Function

We introduce a deformation method of a 3D object based on a key point interpolation. We utilize the deformation method based on RBF [16]. The 3D object is created by adding all the shapes which are calculated by multiplying each key point by RBF (Figure 2). Therefore, let the pairs of key points to be K^0 and K^t where K^t is a vectors of 3D points $(k_0^t, \dots, k_M^t)^T$ of the face F^t of t -th frame, then, K^t can be calculated by a sum of the

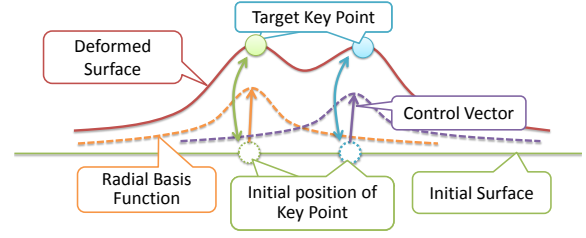


Figure 2: RBF based deformation

multiplication of all $k_i^0 (i \in M)$ by RBF with the control vector $C^t = (c_0^t, \dots, c_M^t)^T$ as follows (Figure 2):

$$K^t = RBF(W(K^0, K^0))C^t \quad (1)$$

where

$$W(K^0, K^0) = \begin{bmatrix} \|k_0^0 - k_0^0\| & \dots & \|k_0^0 - k_M^0\| \\ \vdots & \ddots & \vdots \\ \|k_M^0 - k_0^0\| & \dots & \|k_M^0 - k_M^0\| \end{bmatrix}. \quad (2)$$

The values of the matrix W are the parameters of the RBF according with the distance from each point in K^0 to other points. If the inverse matrix of $RBF(W(K^0, K^0))$ exists, we can solve the equation (1) to find the value of C^t . By using the inverse matrix of $RBF(W(K^0, K^0))$ and the vectors C^t , the key points K^0 are correctly transformed to K^t . Finally, by using the set of key points K^0 and the control vectors C^t , an arbitrary point x is deformed by the following equation.

$$x' = RBF(\hat{W}(x, K^0))C^t \quad (3)$$

x' is a deformed point to the t -th frame. By using this deformation method, we can deform all the points of frame 0 to any frame. In the following sections, we represent this process of the deformation from the face F^0 to F^t as the following equation.

$$\hat{F}^t = D(F^0 \rightarrow F^t) \quad (4)$$

\hat{F}^t is a face which is deformed from F^0 to fit to F^t . And by using the set of key points K^t , we can also make deformations form arbitrary frame number to another frame.

4.1.1 Kernel function

The feature of the kernel function of the RBF is important to deform a 3D object smoothly. In our study, we define the kernel as the Gaussian function.

$$RBF(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x}{\sigma}} \quad (5)$$

The Gaussian function is positive and symmetric. Thus, if the value of σ is the same in each key point, the weighting matrix $RBF(W)$ has an inverse matrix. By the reason, we use the fixed value for the σ .

5 ADDITION OF EXTRA KEY POINTS

We detect and add extra key points to reduce the difference between F^0 and $D^t(F^t \rightarrow F^0)$. To calculate the difference between F^0 and $D^t(F^t \rightarrow F^0)$, we use e_i^t which is a distance between the point p_i^0 in F^0 and its nearest point in $D^t(F^t \rightarrow F^0)$. Since our deformation method transforms the key points to target position and multiply with RBF which decrease the weight according to the distances from the key points, an error becomes large in the area where the key points are sparse (Figure 2 and Equation (3)). To overcome this problem, the area where the error is large is detected and a new key point is added.

5.1 Initial key point in 2D space

Our method requires several numbers (10 - 30) of initial key points on a 2D image. Finally we detect a suitable set of key points automatically. However, since our method evaluates the difference between a standard face and normalized ones to detect the area lacking of key points, the method requires an initial guess of a deformation.

There is a lot of method to detect features of 3D object is proposed[1]. These methods are useful to track rigid movements of the objects. However these methods cannot detect the same positions on a face between two faces. And it is difficult to track the same position of a face with topological transformations (e.g. open a mouth or blink eyes) by using these methods. Therefore we utilize the method proposed by Cao *et al.* [5] for finding facial landmarks from a 2D image of the face. This method can find the facial landmarks with the accuracy over 95%. This accuracy is enough for the initial set of key points for our method. Before the detection process of extra key points, we apply this method to initialize the set of key points K^t .

5.2 Key point candidate selection

Since the shape deformation in our method is calculated by a key point and its interpolation, the error near the key point is naturally small. Therefore, it is a straightforward to find a new key point from the large error area. In our method, we select the candidates for the new key point which satisfies the following conditions:

1. The sum of the errors of neighboring points is larger than the threshold.
2. An error of the key point is larger than neighboring points.

Since the number of 3D points of a face is usually large, comparing the criteria for all the point is impractical, we randomly sample the candidates of key points Kc_k^0 from the F^0 . In our implementation, we sample 4,000 points.

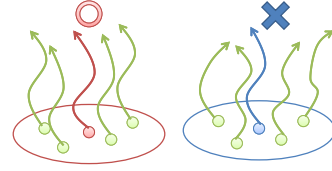


Figure 3: Filtering of candidates of key points. (a)The area where the correlation is high. (b)The area where the correlation is low because of there are outliers.

5.3 Filtering the candidate using temporal consistency

Since selected candidates usually include many outliers because of noise, occlusion and other reasons, filtering is necessarily required. In our method, we check the temporal consistency of neighboring points (Figure 3). The detailed calculation process is as follows:

1. In terms of the candidates of key points Kc_k^0 , find the neighboring points np_j inside the sphere of radius R .
2. The error values of e_k^t and e_j^t ($j \in R$) are calculated. e_k^t is the minimum distance form Kc_k^0 to $D^t(F^t \rightarrow F^0)$ and e_j^t is the distances is the minimum distance form np_j to $D^t(F^t \rightarrow F^0)$.
3. Time series of e_k^t and e_j^t is calculated for all the frames of the face F^t to make a sequence of errors e_k and e_j .
4. The correlation between e_k and e_j are calculated. The higher value of the correlation means that this candidate of a key point Kc_k^0 is placed on the center of a large movement of a face.
5. If the correlation is below the average of all the candidates, the point Kc_k^0 is rejected.

Through this process, we reduce the candidates Kc_k^0 under 40.

5.4 Refinement of the new key points

Since the candidates of key points are selected from the area where the errors are larger than other area, the nearest point in $D^t(F^t \rightarrow F^0)$ form Kc_k^0 is usually the wrong point for deciding positions of the key points in other frames Kc_k^t . Therefore, we utilize the Non-Rigid Registration method which uses the Gaussian Mixture model for representing 3D points for registrations [10] for a local search for the positions of the key points Kc_k^t (Figure 4). The detailed process is as follows:

1. By using $D^t(F^0 \rightarrow F^t)$ (with the current set of the key points), we deform the F^0 for creating "currently tracked" faces in each frame. Then, the Kc_k is also deformed to Kc_k^t by the deformation.

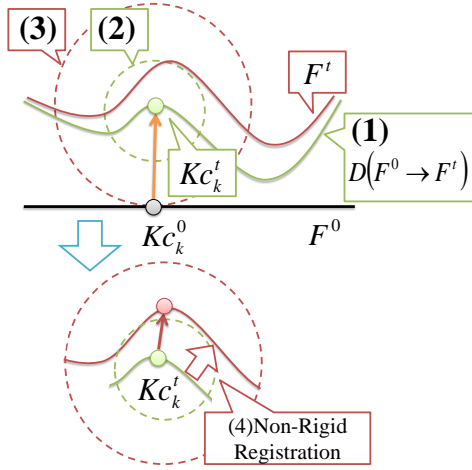


Figure 4: Refinement of key points

2. A surface of $D^t(F^0 \rightarrow F^t)$ inside the sphere of radius R and the center of which is the key points Kc_k^t is selected. The selected surface represents a shape of the current face which contains the errors.
3. Similarly, another surface of F^t inside the sphere of radius $2R$ and the center of which is Kc_k^t is selected. The second surface represents a shape of the original face F^t around the key point.
4. By using the Non-Rigid Registration method, the first surface fits into the second one. Then, the positions of the key points Kc_k^t are also fitted to the F^t .

Through the process, pairs of additional key points Kc_k^0 and Kc_k^t are detected and added to the list of the key points K^t .

5.5 Iterative step for adding key points

We iterate the process of adding key points until the ratio of error is decreased to less than 1% from the previous process. During the iteration process, we also remove the existing key points which have larger error than newly added key points. Initial key points are also removed in the actual process in our experiments.

6 FACIAL TRANSFER METHOD BASED ON THE FACE DEFORMATION

Facial expressions and motions on a face are important presentation in animations. To utilize the captured motions efficiently, the methods transfer from the facial motion of a person to other persons are proposed. The transfer methods are used in the following examples: (a) re-product facial animations of a person by using a motion data of another person. (b) "Stuntman" of the

facial motion. (c) to create animations of virtual characters.

These methods use pairs of key points to define the correspondence between a transfer source shape and a target one. One of the important points of this kind of methods is how to interpolate the difference of both shapes. One of the proposed methods[19] defines a zone of influence and a weight map for interpolating the movement of key points. It is a time consuming process to set the parameters of key points. To solve this problem, we utilize the face space deformation method to transfer facial motions. This method deforms 3D space around a face based on positions of key points. Therefore, the method can be a solution to represent a conversion from a movement of a face to that of other faces. This method consists of following four steps.

- (1) Creating deformations from a source motion of a face to a target one.
- (2) Key point sampling to define correspondences between the source and target faces.
- (3) Detection of weighting values of key points for deformations.
- (4) Creating deformations of the target face.

The details of these are explained in the following sections.

6.1 Deformation from a source motion of a face to a target face

First, we deform a sequence of a facial motion F^t (source face) to fit into a shape of another face T^0 (target face). To define the deformation based on the face space deformation, we use a set of key points described in the section 5.1. Then, the deformations are given by the equation 6.

$$\hat{F}T^t = D^t(F^t \rightarrow T^0) \quad (6)$$

By using the deformation $D^t(F^t \rightarrow T^0)$, each source face F^t is deformed to fit to the target face T^0 . Although the structures of $\hat{F}T^t$ differ to T^0 , the shapes of $\hat{F}T^t$ close to T^0 . Since the T^0 is a static, the remaining difference between T^0 and $\hat{F}T^t$ means the movement of the face in the space of the target face. Then, we define a correspondences between $\hat{F}T^t$ and T^0 .

6.2 Key point detection for a face transfer

Since the number of the points in the T^0 and $\hat{F}T^t$ is too large for defining deformations, our method defines pairs of key points as follows:

- (1) We define a set of key points Kt_i by random sampling from the target face T^0 .

- (2) Another set of key points Kf_i^t which are the nearest points in \hat{F}^t from each point of Kt_i are detected.

The relationships between Kt_i and Kf_i^t are the pairs of key points.

6.3 Weighting values of key points

We define direction vectors V_i from the target face to the source motion based on the pairs of key points Kt_i and Kf_i^t as follows:

$$V_i^t = \frac{Kf_i^t - Kt_i}{\|Kf_i^t - Kt_i\|} \quad (7)$$

Since the difference between T^0 and \hat{F}^t are caused by the movement of the face F^t , these vectors V_i^t represent directions of movement of the face in each key point. However, since the positions of key points Kf_i^t are fitted to the target face (F^t is deformed in the space of another face T^0), the features of the movement of the source face are decreased by the deformations. Then, we introduce weighting values of key points to improve the feature of the movement of F^t . We consider the initial position of the key point Kf_i^0 to calculate the weighting values. Then, the weighting values W_i of key points Kf_i^t are given by following equation 8.

$$W_i^t = \frac{\alpha}{N_k} \|Kf_i^0 - Kf_i^t\| \quad (8)$$

α : a parameter for controlling the strength of transfer effect.

N_k : a number of the key point Kf_i^0

6.4 Deformations of the facial transfer

Finally, we construct deformations of the facial transfer D_{trns}^t . By using the direction vectors of the key points V_i and the weighting values W_i , the deformations D_{trns}^t are given by following equation.

$$D_{trns}^t(Kt_i \rightarrow (W_i^t V_i^t + Kt_i)) = \hat{T}^t \quad (9)$$

\hat{T}^t represents the transfered face from T^t with the pair of key points Kt_i and $W_i^t V_i^t + Kt_i$. Though these four steps mentioned in the beginning of this chapter, the facial transfer method generates a motion of an arbitrary face that reflects motion of another face.

7 RESULTS AND DISCUSSIONS

In this section, we show the results of a facial motion transfer and the efficiency of a key point refinement technique. First, we evaluate the accuracy of deformations by calculating errors between the original faces to the deformed faces. Next, we discuss about the results of our facial transfer method. In our experiment, we use three types of facial motions: "Slap", "Smile" and "Stretch" (Fig. 5). These motions consist of about 100,000 to 200,000 points and 47 to 60 frames. We implement the proposed method on PC with Xeon X5650 processor. It takes about 10 minutes for processing one of the iteration described in section 5.5.

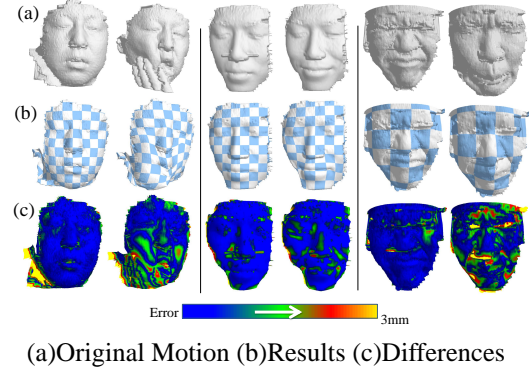


Figure 5: The comparison of the original and captured face.

Table 1: The errors between the original motion and captured motion.

	mean error (mm)		RMSE (mm)	
Name	Initial	Final	Initial	Final
Slap	1.49	1.00	1.89	1.41
Smile	0.64	0.44	0.87	0.66
Stretch	1.23	0.84	1.62	1.22

7.1 Accuracy of the motion capture

In this section we discuss about the error between the original motion and a motion which is deformed from the standard face by using the key points. Table 1 shows the mean errors and RMSE tracked by using the proposed method. The initial error is the mean distance between the face which is deformed with the initial set of key points and the original face. The final error is the mean distance when the recursive process reaches to the convergence condition. The proposed method enables us to minimize the mean error less than 1.0mm on each motion.

Table 2 shows the errors without the key point selection mentioned in the section 5.3 and 5.4 for comparisons. The table shows that the errors are larger than two times those of the proposed method. This result shows that the proposed key point selection method can select the suitable set of key points.

Figure 5(b) and (c) show the deformed motions and the distributions of the errors. This result (b) shows that the proposed method smoothly keeps the continuity of the texture in each frame. The distribution of the error shows that the errors are still remaining around the area where the hand touches the face. Since our method evaluates the motion of key points based on the standard face, it is difficult to detect the motion of an additional object such as a hand. Although there are some errors, our method succeeded in detecting the correspondence between the standard face and the other motions.

Table 2: The errors based on randomly adding the key points.

Name	mean error (mm)	RMSE (mm)
Slap	2.25	2.76
Smile	1.07	1.52
Pinch	1.67	2.09

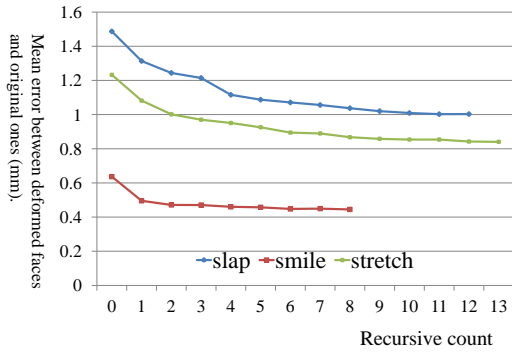


Figure 6: The convergence of the mean error.

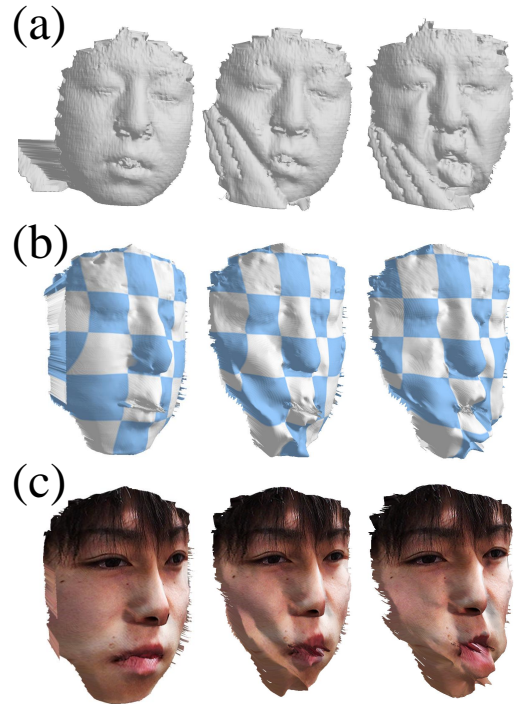
7.2 Results of the facial transfer

Figure 7 shows the results of the proposed facial transfer method. (a) is the original motion (Slap). The left image of (b) is the target shape of a face for applying the facial transfer. The results in (b) and (c) show that the movement of the left cheek and the lip can be transferred to the target face. In the right images of (b) and (c), there are noises around the lip and lower jaw. Since our transfer method randomly samples candidates of key points, key points are placed on the noisy area of the source motion in sometimes. Although the clustering method mentioned in section 5.4 chooses the center of the candidates as a key point, this problem has still happened in this case. To solve this problem, we should apply a constraint function such as the Tevs's facial tracking method[17] for a deformation of a face.

8 CONCLUSIONS

We proposed the key point detection and refinement method for a facial motion which is represented by dense point cloud. We also proposed the facial transfer method base on the key points. The contributions of this paper are as follows:

- (1) We propose the face space deformation which can represent the movement of a face as the RBF based deformations.
- (2) By utilizing the face space deformation, the proposed method can define the suitable pairs of key points to minimize the errors.



(a)Original Motion (b)Transfer target(left) and transferred shape of the face. (c)Transferred motion with a texture.

Figure 7: The comparison of the original motion and the transferred motion.

- (3) In our results, we show that the mean errors between the original motions and tracked motions can reduce less than 1.0mm in all three types of facial motions.
- (4) We also show our facial transfer method can transfer the complex motion on a face to another face.

In the future work, a refinement technique of a surface of a face is needed to generate smoother animations.

9 ACKNOWLEDGMENT

This work was supported in part by the Funding Program for Next Generation World-Leading Researchers(NEXT Program) No.LR030 and Grant-in-Aid for Young Scientists(B) No.25870570 in Japan.

10 REFERENCES

- [1] Bariya, P., Novatnack, J., Schwartz, G., Nishino, K.: 3d geometric scale variability in range images: Features and descriptors. *Int. J. Comput. Vision* **99**(2), 232–255 (2012)
- [2] Belhumeur, P., Jacobs, D., Kriegman, D., Kumar, N.: Localizing parts of faces using a consensus

- of exemplars. In: *Computer Vision and Pattern Recognition (CVPR)*, 2011 IEEE Conference on, pp. 545–552 (2011)
- [3] Bickel, B., Lang, M., Botsch, M., Otaduy, M.A., Gross, M.: Pose-space animation and transfer of facial details. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08*, pp. 57–66. Eurographics Association (2008)
- [4] Bradley, D., Heidrich, W., Popa, T., Sheffer, A.: High resolution passive facial performance capture. *ACM Trans. Graph.* **29**(4), 41:1–41:10 (2010)
- [5] Cao, X., Wei, Y., Wen, F., Sun, J.: Face alignment by explicit shape regression. In: *CVPR*, pp. 2887–2894. IEEE (2012)
- [6] Cosker, D., Roy, S., Rosin, P.L., Marshall, D.: Re-mapping animation parameters between multiple types of facial model. In: *Proceedings of the 3rd international conference on Computer vision/computer graphics collaboration techniques, MIRAGE'07*, pp. 365–376. Springer-Verlag, Berlin, Heidelberg (2007)
- [7] Fratarcangeli, M., Schaerf, M., Forchheimer, R.: Facial motion cloning with radial basis functions in mpeg-4 fba. *Graph. Models* **69**(2), 106–118 (2007)
- [8] Huang, D., De La Torre, F.: Facial action transfer with personalized bilinear regression. In: *Proceedings of the 12th European conference on Computer Vision - Volume Part II, ECCV'12*, pp. 144–158. Springer-Verlag, Berlin, Heidelberg (2012)
- [9] Huang, H., Chai, J., Tong, X., Wu, H.T.: Leveraging motion capture and 3d scanning for high-fidelity facial performance acquisition. *ACM Trans. Graph.* **30**(4), 74:1–74:10 (2011)
- [10] Jian, B., Vemuri, B.C.: Robust point set registration using gaussian mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.* **33**(8), 1633–1645 (2011)
- [11] Moro, A., Mumolo, E., Nolich, M.: Automatic 3d virtual cloning of a speaking human face. In: *Proceedings of the 2010 ACM workshop on Surreal media and virtual cloning, SMVC '10*, pp. 45–50. ACM, New York, NY, USA (2010)
- [12] Pyun, H., Kim, Y., Chae, W., Kang, H.W., Shin, S.Y.: An example-based approach for facial expression cloning. In: *ACM SIGGRAPH 2006 Courses, SIGGRAPH '06*. ACM, New York, NY, USA (2006)
- [13] Sattar, A., Stoiber, N., Segulier, R., Breton, G.: Gamer's facial cloning for online interactive games. *Int. J. Comput. Games Technol.* **2009**, 9:1–9:16 (2009)
- [14] Seol, Y., Lewis, J., Seo, J., Choi, B., Anjyo, K., Noh, J.: Spacetime expression cloning for blend-shapes. *ACM Trans. Graph.* **31**(2), 14:1–14:12 (2012)
- [15] Sibbing, D., Habbecke, M., Kobbelt, L.: Markerless reconstruction and synthesis of dynamic facial expressions. *Comput. Vis. Image Underst.* **115**(5), 668–680 (2011)
- [16] Sumner, R.W., Schmid, J., Pauly, M.: Embedded deformation for shape manipulation. In: *ACM SIGGRAPH 2007 papers, SIGGRAPH '07*. ACM (2007)
- [17] Tevs, A., Berner, A., Wand, M., Ihrke, I., Bokeloh, M., Kerber, J., Seidel, H.P.: Animation cartography –intrinsic reconstruction of shape and motion. *ACM Trans. Graph.* **31**(2), 12:1–12:15 (2012)
- [18] Valgaerts, L., Wu, C., Bruhn, A., Seidel, H.P., Theobalt, C.: Lightweight binocular facial performance capture under uncontrolled lighting. *ACM Trans. Graph.* **31**(6), 187:1–187:11 (2012)
- [19] Vieira, R.C.C., Vidal, C., Cavalcante-Neto, J.B.: Expression cloning based on anthropometric proportions and deformations by motion of spherical influence zones. In: *SBC Journal on 3D Interactive Systems*, pp. 14–22 (2011)
- [20] Weise, T., Bouaziz, S., Li, H., Pauly, M.: Realtime performance-based facial animation. In: *ACM SIGGRAPH 2011 papers, SIGGRAPH '11*, pp. 77:1–77:10. ACM, New York, NY, USA (2011)
- [21] Zhang, W., Wang, Q., Tang, X.: Real time feature based 3-d deformable face tracking. In: *Proceedings of the 10th European Conference on Computer Vision: Part II, ECCV '08*, pp. 720–732. Springer-Verlag, Berlin, Heidelberg (2008)

(Semi) regular tetrahedral tilings

Alexej Kolcun

Institute of Geonics, Czech Academy of Sciences
Studentská 1768
708 00 Ostrava, Czech Republic
alexej.kolcun@ugn.cas.cz

ABSTRACT

Triangulation 2D and 3D methods represent an important part of numerical modeling process (e.g. FEM). In many applications it is suitable to use triangulations with a regular structure. Moreover, decomposition with a limited number of different types of tetrahedra enables to reduce the storage and computational demands in the whole modeling process. The submitted contribution presents an overview of possible 3D decompositions using one tetrahedron (regular tilings). These decompositions are confronted with decompositions using six different types of tetrahedra (orthogonal structured tilings). Considering the shape expressivity of particular methods, the paper presents a structure of possible decompositions and comparison of storage demands of the used decompositions.

Keywords

tetrahedron, regular tessellation, voxel grid, conform decomposition.

1. INTRODUCTION

The tasks connected with space decomposition have been presented in various research areas for a long time, and they are connected with Hilbert's 18th problem [H], [G 1980]. The whole raster graphic concept can be seen from this point of view. A different raster concept, based on regular hexagonal mesh, is analyzed e.g. in [M 2005].

Within the numerical methods development, space discretization has become an important tool of shape expressivity. For example, in the finite element modeling (FEM), the domain of interest is decomposed to simple polyhedral elements. In the simplest case, the triangles for 2D tasks and tetrahedra for 3D tasks are used. Nowadays, there is a wide range of generators used for decomposition (meshes), e.g. [E 2001], [F 2000].

Surprisingly, we may expect that with HW development the role of regular meshes for physically based modeling will grow. For example, CT

technologies of 3D scanning give us information about geometry in a regular rectangular grid. Thus, the originally demanding preparation of the analyzed domain geometry is possible to fully automatize in a very simple way, [P 2003], [A 2007].

Advantages of this approach:

1. simplicity,
2. velocity,
3. relatively small storage demands (hereby described geometry contains a huge amount of grid vertices, but it is not necessary to keep their coordinates – these can be simply calculated),
4. regular structure of the equations system which must be solved.

The results of this approach significantly exceed its disadvantages:

1. From the numerical model viewpoint, the discretization error – aliasing – of boundaries of particular domain areas has only a very local character [A 2007].
2. Moreover, this error may be eliminated by the pixel/voxel partitioning; either from the geometry viewpoint – Fig. 1, [L 2013] or from the numerical solution viewpoint – Fig. 2. Even though this method is only partial, it is very effective in many cases.
3. Using Marching Cubes and Marching Tetrahedra methods (originally developed

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

directly for these rectangular grids), it is possible to eliminate effectively this discretization error, e.g. [P 2005].

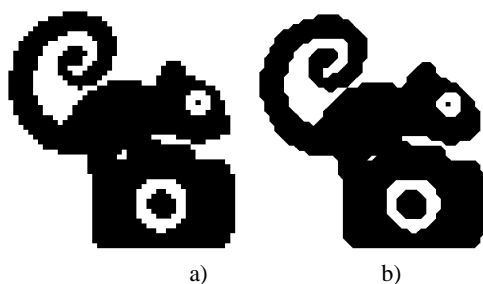


Figure 1 a) Raster representation and b) partitioned raster representation [L2013].

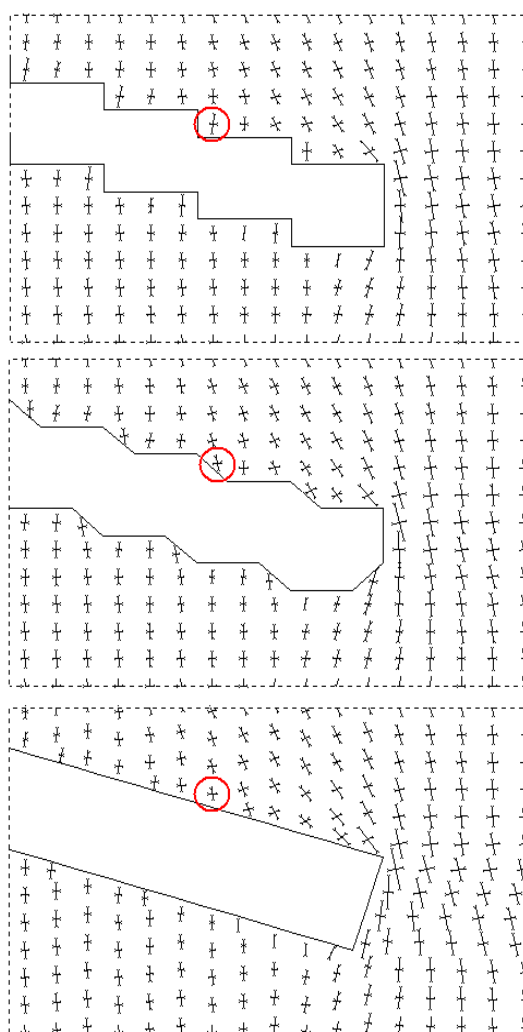


Figure 2 Local character of the influence of the domain boundary aliasing to the stress tensor.

Next approach of regular rasterization [L 2013] can be based on equilateral triangles – Fig. 3a): regular rectangular mesh (black lines) is deformed and proper diagonals (gray lines) are added. In this case, the major part of triangles (all except for the

boundary ones) is equilateral. Fig. 3b) gives the example of such rasterization. While in the concept of the partitioning from Fig. 1b) the choice of the diagonals depends on material (color) distribution in the raster grid, the partitioning in the concept from Fig. 3a) is regular – ‘zig-zag’.

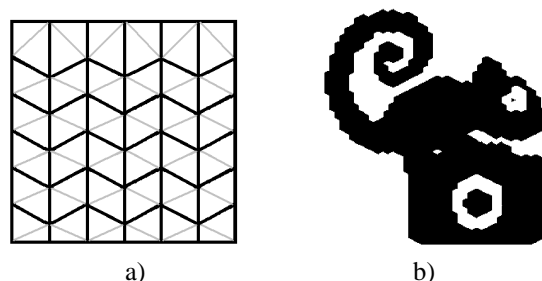


Figure 3 a) Generalized rasterization based on a ‘deformed’ regular grid (black lines), supplemented with diagonals (gray lines), b) example of rasterized image [L2013].

The paper presents an overview of all known space decomposition methods to identical tetrahedra (regular tetrahedral tiling). This mechanism influences the result of the Marching Cubes (Tetrahedra) method.

While Sommerville decompositions (Section 2.1) are relatively well known and widely used in the computer modeling area, Goldberg decompositions (Section 2.2) are considerably less known among the computer graphic community. Mutual dependences of these decompositions are mentioned.

The method of orthogonal structured tilings (decomposition to several different tetrahedra) is also described.

A comparison of the shape expressivity of mentioned methods is presented.

2. REGULAR TETRAHEDRAL TILINGS

At present, two approaches of regular tetrahedral tilings are known [G 1978], [S 1981].

1. decompositions obtained by partitioning of a regular rectangular grid using only one type of tetrahedral element (Sommerville),
2. decompositions obtained by partitioning of a 3-sided prism (Goldberg).

2.1 Regular rectangular grid partitioning

Let us consider a regular rectangular grid (voxel grid). Its basic volume element is an orthogonal prism (brick). There are five partitioning tetrahedra obtained by its decomposition. The basic one is tetrahedron I. (Fig. 4), where vertex D is the centre of the cube, C is the centre of the right side.

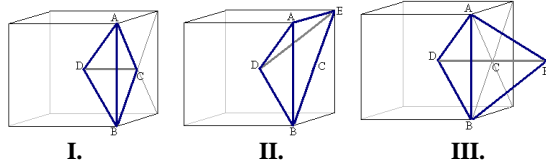


Figure 4 Tiling tetrahedra, part 1.

The tetrahedron II (Fig. 4) is a union of two tetrahedra of the type I with common face ACD. The tetrahedron III is a union of the two tetrahedra of the type I with common face ABC. Table 1 shows the mutual rate of edges of the described tetrahedra.

I. ABDC	AB 2	BD $\sqrt{3}$	AD $\sqrt{3}$	AC $\sqrt{2}$	BC $\sqrt{2}$	DC 1
II. ABDE	AB 2	BD $\sqrt{3}$	AD $\sqrt{3}$	AE 2	BE $2\sqrt{2}$	DE $\sqrt{3}$
III. ABDF	AB 2	BD $\sqrt{3}$	AD $\sqrt{3}$	AF $\sqrt{3}$	BF $\sqrt{3}$	DF 2
IV. ABDH	AB 2	BD $\sqrt{3}$	AD $\sqrt{3}$	AH $\sqrt{5}/2$	BH $\sqrt{5}/2$	DH $\sqrt{5}/2$
V. ABJF	AB 2	BJ $\sqrt{11}/2$	AJ $\sqrt{3}/2$	AF $\sqrt{3}$	BF $\sqrt{3}$	JF $\sqrt{11}/2$

Table 1 Lengths of edges of the tiling tetrahedra for a rectangular grid partitioning.

Note that the tetrahedron II is asymmetrical and in this case, the decomposition has to contain mutually symmetric pairs of tetrahedra. We say that such a regular tiling is asymmetrical.

Next two types of partitioning tetrahedra are derived from the type III:

Let us consider the node G – the centre of the edge AB and the node C – the centre of the edge DF (Fig. 4 III'). We abbreviate H – the centre of the segment CG. It is obvious that H is the centre of the sphere circumscribed to the tetrahedron ABDF. Because of the fact that the faces of ABDF – III (Fig. 4, Tab. 1) are identical, using the vertex H we can decompose this tetrahedron to four identical tetrahedra ABDH, BAFH, DFAH, and FDBH. Thus the ABDH tetrahedron generates a symmetrical tiling (Fig. 5 IV.).

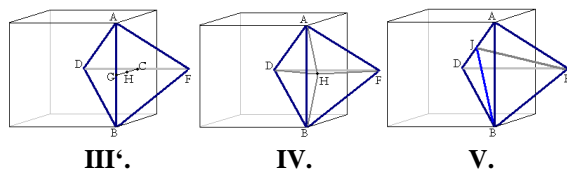


Figure 5 Tiling tetrahedra, part 2.

Let J be the centre of the edge AD. Then the tetrahedra ABFJ and DFBJ are symmetrical, i.e.

tetrahedron ABFJ generates an asymmetrical tiling (Fig. 5 V.).

2.2 Decompositions from 3-sided prism

Let us consider a right prism whose normal section is an equilateral triangle of edge e . On the edges of the prism we generate vertices by „gradual rolling up“ (red polyline) with given shift h (Fig. 6).

The tetrahedron ABCD – VI. obviously generates a tiling: let us consider neighbor tetrahedra ABCD and BCDB'. Each of them contains three edges of length a – red, (AB,BC,CD, BC,CD,DB' respectively), two edges of length b – blue, (AC,BD, BD,CB' respectively), one edge of length $3h$ – black AD, BB' respectively). Moreover, both tetrahedra ABCD and BCDB' have the same orientation.

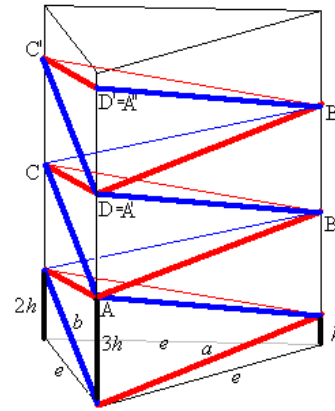


Figure 6 Basic partition of 3-sided prism – VI.

Since the ratio h/e is arbitrary, there is a continuous infinity of tiling tetrahedra of this type where

$$a = \sqrt{e^2 + h^2}, \quad (1)$$

$$b = \sqrt{e^2 + 4h^2}. \quad (2)$$

Another two tiling tetrahedra – VII., VIII., (Fig. 7) can be derived in the following way:

vertex E is the centre of the edge AD, and so the triangle BCE is isosceles, i.e. $BE=CE=c$, where

$$c = \sqrt{e^2 + \left(\frac{h}{2}\right)^2} = \frac{\sqrt{4e^2 + h^2}}{2}. \quad (3)$$

Similarly the vertex F is the centre of the edge BC and so the triangle ADF is isosceles, i.e. $AF=DF=d$. Moreover, angle $\angle BFD$ is right one, so

$$d = \sqrt{b^2 - \left(\frac{a}{2}\right)^2} = \sqrt{e^2 + 4h^2 - \frac{e^2 + h^2}{4}} = \frac{\sqrt{3e^2 + 15h^2}}{2}. \quad (4)$$

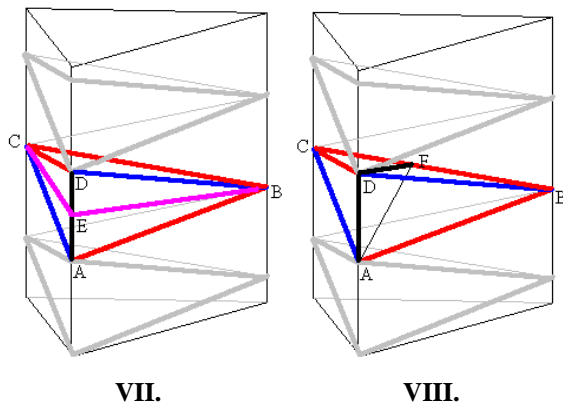


Figure 7 Tiling tetrahedra for 3-sided prism.

Table 2 shows the lengths of edges of partitioning tetrahedra for a 3-sided prism.

VI. ABCD	AB <i>a</i>	BC <i>a</i>	AC <i>b</i>	AD <i>3h</i>	BD <i>b</i>	CD <i>a</i>
VII. ABCE	AB <i>a</i>	BC <i>a</i>	AC <i>b</i>	AE <i>1,5h</i>	BE <i>c</i>	CE <i>c</i>
VIII. ABFD	AB <i>a</i>	BF <i>a</i>	AF <i>d</i>	AD <i>3h</i>	BD <i>b</i>	FD <i>d</i>

Table 2 Lengths of edges of the tiling tetrahedra for a 3-sided prism partitioning.

Lemma 1. The below described relations between tiling tetrahedra are valid:

- tetrahedron III. is a special case of the trahedron VI.,
- tetrahedron II. is a special case of the tetrahedron VII.,
- tetrahedron V. is a special case of the tetrahedron VII., VIII., respectively.

Proof: We shall prove relation a).

Let us consider tetrahedron VI. with

$$8h^2 = e^2.$$

According to (1), (2)

$$a = 3h, b = 2\sqrt{3}h, .$$

So, we have obtained tetrahedron III. – Tab. 1.

The rest of relations can be proved in similar way. More detailed proof in [G 1978]. *q.e.d.*

So far, there are not known any other regular tilings; on the other hand, their absence has not been proved yet [G 1978].

2.3 Decomposition conformity

Within the context of numerical modeling, the possibility of continuous interpolation is usually required. This is the reason why we suppose *conformity* of the used decomposition, i.e. the neighbor elements share either just one vertex, or the whole edge, or the whole face. This fact is automatically guaranteed by the above-described tilings, for the cases I. – V. excluding the decompositions based on tetrahedra II. For conformity of the tilings based on tetrahedron II, it is required that the neighbor voxels must be partitioned by the same diagonal (which is quite a natural assumption).

In the case of tetrahedra VI. – VIII., the „gradual rolling up“ in the neighbor 3-sided prism must be realized in the way of the mirror symmetry. Generally, if the tetrahedron VI. is asymmetric, the corresponding tilings are asymmetric too.

3. ORTHOHONAL STRUCTURED TILINGS

Let us show another way of decomposition. If we extenuate the condition of the tiling, i.e. we consider more than one type of tetrahedra (but the finite number), it is required that the tetrahedra vertices be the vertices created just from the original vertices from the orthogonal grid (*inscribed tetrahedra*). This decomposition is called *orthogonal structured one*.

The following relations are valid (the class of elements, decompositions respectively, is the set of elements, decompositions respectively, which differ only by the angle of rotation).

Lemma 2. There are 5 tetrahedra classes creating an orthogonal structured tiling (Fig. 8).

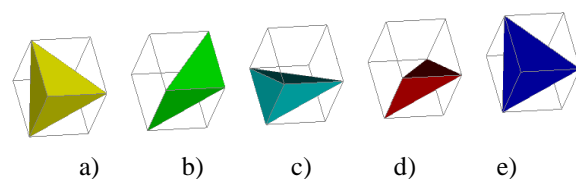


Figure 8 Tetrahedra of an orthogonal structured tiling.

The proof is simple; we obtain it by a detailed analysis of all four-element subsets of the orthogonal brick. *q.e.d.*

Corollary: there are 58 different tetrahedra inscribed into the cube.

Lemma 3. There is one class of the conform decomposition to 5 tetrahedra and 5 classes if the

conform decompositions to 6 tetrahedra (Fig. 8 – 9). [A 2003], [K 1999].

The proof results from the following facts.

1. The analysis of the conformal 5-element decompositions is trivial.
2. Conformal decomposition to 6-tetrahedra contains exactly one body diagonal.
3. The decomposition classes are uniquely determined by replacing the pair of tetrahedra b)c) from Fig. 8 with the pair a)d) – see Fig. 9a)-b). Similarly, we can replace other two pairs of tetrahedra b)c). The last type of decomposition is from Fig. 9e).

q.e.d.

Figures 9a) and e) show the most important decompositions often used within FEM modeling. Their important feature is that in both cases all three pairs of diagonals on opposite faces are mutually parallel.

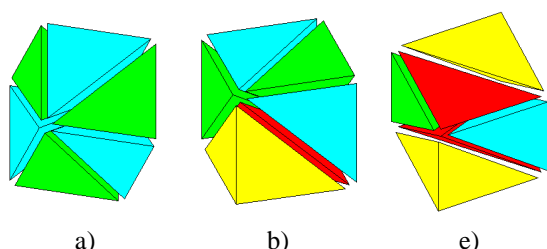


Figure 9 Examples of orthogonal structured tilings.

Figure 10 shows dual representation of all possible classes of conformal orthogonal structured tilings: tetrahedron is represented as a node, nodes are connected with the edge iff the tetrahedra are neighbor. The colors of tetrahedra from Figure 8 and nodes from Figure 10 mutually correspond. The abbreviation of tetrahedra from Fig. 9 and dual representations from Fig. 10 mutually correspond, i.e. the decomposition in Fig.9e corresponds to the representation in Fig. 10e).

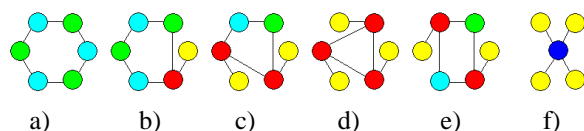


Figure 10 Dual representation of conform decompositions.

Lemma 4. There are 72 decompositions to 6 tetrahedra and two decompositions to 5 tetrahedra [K 1999].

The proof results from the analysis of rotating symmetries of particular classes.

Polyhedra from Fig 11 represent examples of well known Schönhardt's polyhedra, for which there exists no conformal decomposition (e.g. [R 2005]).

The diagonal configurations from Fig. 11 may be characterized as follows: each vertex is incident to the limit of two diagonals and

- a) exactly two pairs of diagonals on the opposite sides are skew,
- b) all three pairs of diagonals on the opposite sides are skew.

Lemma 5 and 6 give finer results than [R 2005].

Lemma 5. For configuration of the surface diagonals from Figure 11a) there are nonconform decompositions only.

Lemma 6. For configuration of the surface diagonals from Figure 11b) there are no decompositions.

The proof of Lemmas 5 and 6 [K 1999] results from the fact, that the nonconform decomposition contains exactly two body diagonals.

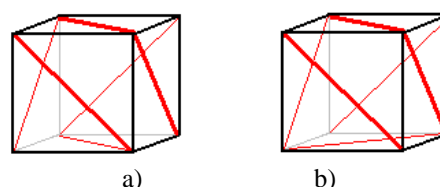


Figure 11 The surface diagonals configuration in which a) there are the nonconform decompositions only b) there is no decomposition.

Importance of last two Lemmas consists in conform decomposition algorithm:

1. Rectangular regular grid is generated.
2. Facial and space diagonals due to prescribed geometry are added.
3. Rest of diagonals is added – avoiding the configurations from the Fig. 11.

So far, there are not known any better algorithms for conform decompositions generation for general prescribed geometry.

Nonconformity decomposition can be solved effectively when the orthogonality of the initial grid is weakened [K 2002].

4. SHAPE EXPRESSIVITY OF THE DECOMPOSITIONS

Let us consider a regular orthogonal grid with n vertices in each direction.

Table 3 shows the dependences of the number of vertices and the tetrahedra.

	Number of vertices	Number of tetrahedra
I.	$n^3 + (4n-1)(n-1)^2$	$24(n-1)^3$

II.	$n^3 + (n-1)^3$	$12(n-1)^3$
III.	$n^3 + (n-1)^3$	$12(n-1)^3$
IV.	$n^3 + 13(n-1)^3$	$48(n-1)^3$
V.	$n^3 + 9(n-1)^3$	$24(n-1)^3$

Table 3: The vertices and tetrahedra in the decomposed voxel grid with n^3 vertices.

On the other hand, all the structured decompositions to 6-tetrahedra keep their original number of vertices n^3 . The number of elements is $6(n-1)^3$.

When editing the primary voxel grid, it is necessary not to be limited by the selection of the surface diagonals on the bricks. Obviously, each of the mentioned decompositions I. – V. allows to select the diagonals on the common face of bricks independently. To be more exact, all the decompositions, except for II, contain both diagonals on each side. In the case of the decomposition II, it is possible to select the face diagonals independently. Here the decomposition is directly determined by the surface diagonals selection.

From this point of view, it is meaningful to compare the orthogonal structured tiling (from Section 3) with the decomposition II only. Table 4 shows an overview of the results.

Decomposition	Number of vertices	Number of tetrahedra	Number of decompositions	The surface diagonals limits
II.	$\approx 2N$	$\approx 12N$	64	No
Struct.	N	$\approx 6N$	72	Yes

Table 4: Global characteristics of the orthogonal structured tilings and tiling II.

5. CONCLUSIONS

The submitted paper describes various approaches to generation of tetrahedral decompositions with a fixedly determined structure. Apart from the commonly used Sommerville decompositions based on a regular orthogonal grid, there are also analyzed Goldberg decompositions, which enable to use a more general discretization grid. Moreover, for the initial orthogonal grid there is also analyzed a more general decomposition to six tetrahedra, using a pentad of different tetrahedra. The basic properties of these decompositions are formulated and it is shown (Tab. 4) how extenuating the conditions may increase the possibilities of expression of various shapes.

ACKNOWLEDGEMENTS

This work was supported by the projects

- RVO: 68145535,
- University of Ostrava, SP13/PRF/2013

REFERENCES

- [A 2003] Apel, T., Duvelmeyer, N.: Transformation of Hexahedral FE-Mesh into Tetrahedral Meshes according to Quality Criteria, *Computing* 71 (2003), pp 293-304.
- [A 2007] Arbenz, P., Flaig, C.: On Smoothing in Voxel Based Finite Element Analysis of Trabecular Bone. In: LSSC2007 proc. (Lirkov, I., Margenov, S., Wasniewski J. eds.), *Lecture Notes in Computer Science* 4818, Springer 2008, pp. 69-77.
- [E 2001] Edelsbrunner, H.: *Geometry and Topology for Mesh Generation*, Cambridge University Press 2001.
- [F 2000] Frey, P.J., George, P.L.: *Mesh generation: Application to Finite Elements*, Hermes Science 2000.
- [G 1978] Goldberg M.: *Three Infinite Families of Tetrahedral Space/Fillers*, *Journal of Combinatorial Theory (A)* 16, 1978, pp.348-354.
- [G 1980] Grunbaum, B., Shepard, G.C.: Tilings with Congruent Tiles. *Bulletin of American Math. Soc.* Vol. 3, No.3, November 1980. pp. 951-973.
- [H] Hilbert's problems, http://en.wikipedia.org/wiki/Hilbert%27s_problems
- [K 1999] Kolcun, A.: The Quality of Meshes and FEM Computations, In WSCG proc. 1999 (V. Skala ed.), pp. 100-105.
- [K 2002] Kolcun, A.: Non-Conformity Problem in 3D Grid Decomposition, *Journal of WSCG*, Vol 10, No.1,(2002), pp. 249-254.
- [L 2013] Lubojacký, J.: *Basic algorithms of rasterization for generalized raster lattice* (in czech), Diploma Thesis, University of Ostrava, 2013.
- [M 2005] Middleton, M., Sivaswamy, J.: *Hexagonal Image processing*. Springer 2005.
- [P 2003] Práger, M.: *On a Construction of Fast Direct Solvers*, *Applications of Mathematics*, Vol. 48, No 3, 2003, pp. 225-236.
- [P 2005] Patera, J., Skala, V.: Centered Cubic Lattice Method Comparison. In. *Proc of Algorithm conf.* 2005, pp. 1-10.
- [R 2005] Rambau, J.: On a Generalization of Schonhardt's Polyhedron, *Combinatorial and Computational Geometry*, Vol. 52(2005), pp. 501-516.
- [S 1981] Senechal, M.: *Which tetrahedra Fill Space?*, *Mathematical Magazine*, Vol. 54, No. 5, 1981, pp. 227-243.

The 3-Point Method: A Fast, Accurate and Robust Solution to Vanishing Point Estimation

Vinod Saini, Shripad Gade, Mritunjay Prasad, Saurabh Chatterjee

Department of Aerospace Engineering,
Indian Institute of Technology, Bombay,
Mumbai, India-400076

{saini.vinod, shripad.gade, mritunjay, saurabhsauroc}@iitb.ac.in

ABSTRACT

Vanishing points can provide information about the 3D world and hence are of great interest for machine vision applications. In this paper, we present a single point perspectivity based method for robust and accurate estimation of Vanishing Points (VPs). It utilizes location of 3 collinear points in image space and their distance ratio in the world frame for VP estimation. We present an algebraic derivation for the proposed 3-Point (3-P) method. It provides us a non-iterative, closed-form solution. The 3-P results are compared with ground truth of VP and it is shown to be accurate. Its robustness to point selection and image noise is proved through extensive simulations. Computational time requirement for 3-P method is shown to be much less than least squares based method. The 3-P method is extremely useful for accurate VP estimation in structured and well-defined environments.

Keywords

Vanishing Points, Point Perspectivity, Length Ratio, Camera Calibration, Cross Ratio

1 INTRODUCTION

A family of parallel lines projected on a plane under the pin-hole camera model will ideally intersect in a common point. This point is known as the Vanishing Point. VP's formed by families of coplanar parallel lines are collinear and the line is known as the Vanishing Line. VPs and Vanishing Lines for an image of a cube are shown in Fig. 1.

Development of computational techniques and ever-growing requirement of extracting information from image have led to a spurt in the field of image analysis in recent years. Vanishing points have myriad applications including camera calibration, robotic navigation, 3D reconstruction, pose estimation, augmented reality etc. VP's have been extensively used for camera calibration. [7], [9], [10], [11] and [12] use VPs for camera calibration. Image reconstruction problem in [13] and [14] use VP's for extraction of 3D coordinates of points. [15] uses parallel lines in the environment and corresponding VP's for steering a robot. [16] uses vanishing points and vanishing lines for pose estimation of UAV's in indoor flights.

Several methods have been proposed to estimate VP's. [3] uses J linkage based algorithm for vanishing point estimation in man-made environments. [4] proposes a new framework for line based geometric analysis and VP estimation of Manhattan scenes. [5] uses accurately localised edges that are obtained through edge pixels and does not require fitting of lines. It uses fewer but more accurate lines for estimating VPs. [8] relies line extraction using Hough transform and then on voting in the vanishing point space to estimate VP. All of the above mentioned methods claim to be accurate for VP estimation in architectural environments e.g. Manhattan scenes.

Vanishing points are extremely important in computer vision and the accuracy of VP estimation directly influences the performance of the said application. Additionally, since there may be a need to compute vanishing points multiple times, a computationally efficient method is the need of the day. Camera calibration efforts require accurate VP estimates Calibration targets are well defined structured objects and this information about them can be exploited to suit our needs. Fig. 2 shows few calibration patterns used in different algorithms. [1] proposes two length ratio based methods. First requires evaluation of 1D projective transformation and the direction of lines to compute VP. Second detects VP using geometric construction. Here we propose a length-ratio based fast and accurate VP estimation method. We use three collinear points and their distance ratio in world frame to compute VP location.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

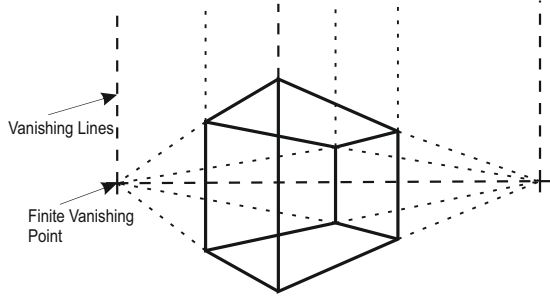


Figure 1: Vanishing Points and Vanishing Line

Section 2 discusses camera calibration preliminaries and least squares vanishing point detection method. Section 3 talks about the 3-P method and its formal derivation. Simulation and experimental results are discussed in section 4.

2 PRELIMINARIES

2.1 Camera Model

Pin-hole model is based on the principle of collinearity, where each point in the world space can be mapped by a straight line to the image plane through the camera center. This kind of central projective transform is called "Perspective". Fig. 3 shows projection of a line on image plane. A Pin-hole camera model has been used here ([1], [2]). Any point P (coordinates given by \mathbf{X}) can be mapped to a point p (coordinates given by \mathbf{x}) in the image plane. The overall transform can be expressed as,

$$\mathbf{x} = \mathbb{P}\mathbf{X} \quad (1)$$

The overall transformation matrix, \mathbb{P} is obtained by multiplying the extrinsic calibration matrix by intrinsic calibration matrix. \mathbb{P} can be expressed as,

$$\mathbf{x} = \mathbb{K} R [I \mid -\tilde{C}] \mathbf{X} = \mathbb{K} \mathbb{M} \mathbf{X} = \mathbb{P} \mathbf{X} \quad (2)$$

Parameters that are solely dependent on camera are called Intrinsic parameters. Principal point, skew, aspect ratio and focal length together form the intrinsic camera calibration matrix (\mathbb{K}). Extrinsic parameters of a camera include the rotation and translation of camera with respect to the world frame. Rotation matrix (\mathbb{R}) and translation vectors (\tilde{C}) together form the extrinsic camera calibration matrix (\mathbb{M}).

2.2 Least Squares VP estimation

Let us consider a set of n parallel lines in 3D. Ideally their mappings in the image plane will intersect at a point (the VP). Due to noise and other errors they will intersect at different points. The maximum number of

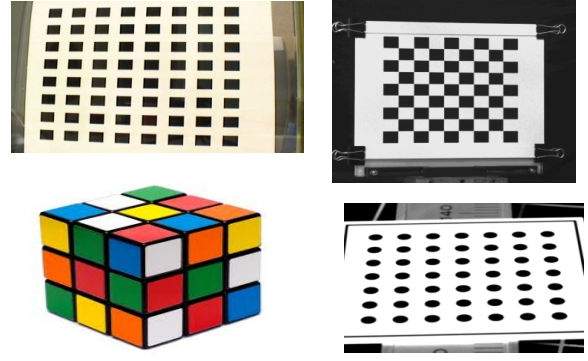


Figure 2: Patterns used to get VPs

intersections that can be found out are nC_2 . The aim is to estimate a point that has least perpendicular distance from the n lines. The methodology can be divided into two sub-tasks,

1. Extracting Lines from the image.
2. Finding Least Squares solution.

Extracting Lines Lines can be extracted from the image using various image processing techniques. In our approach we extract control point locations from the image (see Figure 8). A least squares fit straight line is drawn to minimize perpendicular distance of m points from the line. If (x_i, y_i) are the locations of the m points that lie on a straight line, the slope (slope) and intercept (c) of the line are given by,

$$\begin{bmatrix} c \\ slope \end{bmatrix} = \begin{bmatrix} m & \sum_{i=1}^m x_i \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m x_i y_i \end{bmatrix} \quad (3)$$

Least Squares solution Once the line information is extracted from the image, the only hurdle in estimating the vanishing point using least-squares is finding the intersection of lines. For any two points v_i and v_j , the line passing through them can be expressed as,

$$L_{ij} = v_i \times v_j$$

If m lines given by $L_i (i=1,2,\dots,m)$ intersect in a point V , then the coordinates of V are given by,

$$\begin{bmatrix} L_1^T \\ \vdots \\ L_m^T \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = 0 \quad (4)$$

3 THE 3-POINT METHOD

Parallel lines appear to intersect at a point in perspective view. In an image this perspectivity is introduced due the camera parameters and its orientation. Several methods have been proposed in literature to measure perspectivity. In [1] this perspectivity is measured

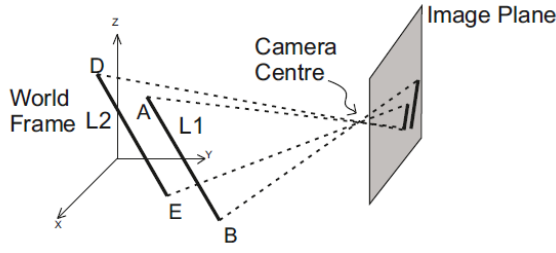


Figure 3: Pin-hole camera model

through the evaluation of 1D projective transformation. The main idea behind our method is that we can get a sense of this camera perspectivity through the three collinear points from the image and their length ratio in the world frame.

Let us consider a family of parallel lines represented by \mathcal{F} . These lines are projected onto an image plane through a pin-hole camera model. Let this set be called as \mathbb{F} . Now, if we select two points lying on any line f ($f \in \mathbb{F}$); we can write the equation of that line f in image space. Equation of line f and the length ratio (given by three collinear points) in the world frame will provide information about perspectivity along f . This will enable us to map any point on line f ($f \in \mathcal{F}$) onto its image f ($f \in \mathbb{F}$). Any point on line f at infinite distance when mapped under pin-hole camera model will map onto VP.

The advantage of this method over the length ratio method is that it does not require us to compute homography (projective transform) and perform intensive computations. It provides us with a closed-form solution and is computationally efficient. Line detection and clustering is not required in this method. This reduces computational load significantly and altogether eliminates line detection and clustering errors. Also this method can be made to utilize information from a small region in the image, thereby reducing errors due to defocusing of certain parts of image.

3.1 Derivation

Let us consider three collinear points $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$ and $C(x_3, y_3, z_3)$. Let, the line that passes through them be called L_1 as shown in Fig. 3.

Distance Ratio: The distance ratio for three collinear points is given by (see Fig. 4),

$$\Gamma = \frac{d_{AC}}{d_{AB}} = \frac{\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2}}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}} \quad (5)$$

Camera Model: We use a pin-hole camera model with projection matrix \mathbb{P} . Let the images of A, B and C

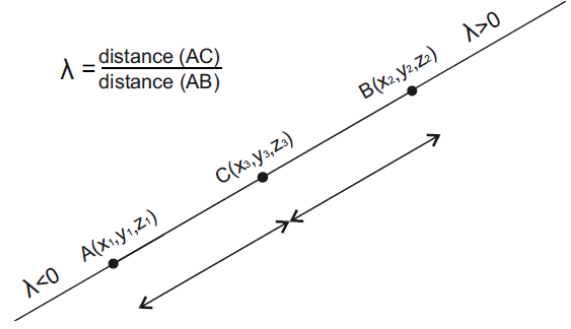


Figure 4: Length Ratio for 3-P method

be called $A'(u_1, v_1, 1)$, $B'(u_2, v_2, 1)$ and $C'(u_3, v_3, 1)$ respectively.

$$A' = \mathbb{P}A, B' = \mathbb{P}B, \text{ and } C' = \mathbb{P}C \quad (6)$$

Line L_1 : Equation of line L_1 (see Fig. 3) can be written in the two point form (using points A and B) as follows,

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1} = \lambda \quad (7)$$

or

$$\begin{aligned} x &= x_1 + \lambda(x_2 - x_1) \\ y &= y_1 + \lambda(y_2 - y_1) \\ z &= z_1 + \lambda(z_2 - z_1) \end{aligned} \quad (8)$$

Now, if we substitute coordinates of point C in Eq. (8) and use the expression in Eq. (5) we can easily conclude that Γ and λ are equal.

Coordinates of C' , for a known distance ratio, can be expressed as,

$$\begin{aligned} \begin{pmatrix} w_3 u_3 \\ w_3 v_3 \\ w_3 \end{pmatrix} &= \mathbb{P} \begin{pmatrix} x_1 + \lambda(x_2 - x_1) \\ y_1 + \lambda(y_2 - y_1) \\ z_1 + \lambda(z_2 - z_1) \\ 1 \end{pmatrix} \\ &= \mathbb{P} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} + \lambda \mathbb{P} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} w_1 u_1 \\ w_1 v_1 \\ w_1 \end{pmatrix} + \lambda \begin{pmatrix} w_2 u_2 - w_1 u_1 \\ w_2 v_2 - w_1 v_1 \\ w_2 - w_1 \end{pmatrix} \end{aligned} \quad (9)$$

Simplifying the above equation we get C' as,

$$(u_3, v_3) = \left(\frac{u_1 + \lambda(\alpha u_2 - u_1)}{1 + \lambda(\alpha - 1)}, \frac{v_1 + \lambda(\alpha v_2 - v_1)}{1 + \lambda(\alpha - 1)} \right) \quad (10)$$

where, α is defined as $\frac{w_2}{w_1}$.

For any point $D(x, y, z)$ lying on line L_1 , and the image $D'(u, v, 1)$ are related by $D' = \mathbb{P}D$. The coordinates of D' are obtained by Eq. 10, and expressed as,

$$(u, v) = \left(\frac{u_1 + \lambda'(\alpha u_2 - u_1)}{1 + \lambda'(\alpha - 1)}, \frac{v_1 + \lambda'(\alpha v_2 - v_1)}{1 + \lambda'(\alpha - 1)} \right) \quad (11)$$

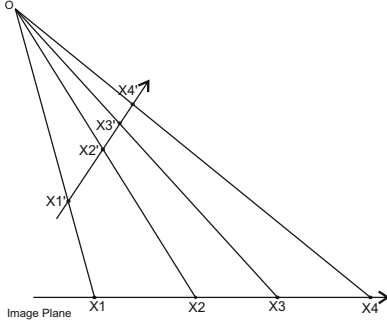


Figure 5: Cross Ratio

In homogenous coordinates, (wu, wv, w) and $(u, v, 1)$ represent the same point. The factor w is merely a scaling quantity. The parameter α is defined as the ratio of scaling factors for two different points. It thereby provides wisdom about perspectivity. α can be evaluated from Eq. 10.

$$\alpha = \frac{w_2}{w_1} = \frac{(u_3 - u_1)(\lambda - 1)}{(u_3 - u_2)\lambda} \quad (12)$$

The vanishing point is the image of a point lying at infinity on line L1. This point (let us say is D) in the world frame will have a distance ratio, $(\lambda' = \frac{d_{A,D}}{d_{A,B}})$ of ∞ . To get the VP, we substitute this value of λ' in Eq. 11. Through algebraic manipulation, we get,

$$(VP_x, VP_y) = \left(\frac{\alpha u_2 - u_1}{\alpha - 1}, \frac{\alpha v_2 - v_1}{\alpha - 1} \right) \quad (13)$$

An interesting phenomenon can be observed if we consider an image with zero perspective. For such an image α is equal to 1 since the scaling factors will be same for both the points. VP for such an image will be located at ∞ (Eq. 13). This is to be expected, since VPs arise only due to perspective in the image.

3.2 Proof by Invariance

Property: The cross ratio (χ) is invariant under projective transformation. χ is expressed as,

$$\chi = \frac{d_{13}d_{24}}{d_{12}d_{34}} \quad (14)$$

where $d_{i,j}$ represents distance between points i, j as shown in Fig. 5.

In our current formulation let us assume that a fourth point V is lying on line L1 at an infinite distance along with A, B and C. V' projected on the image plane from V, will represent the vanishing point. Using Eq. 5, cross ratio in world frame is given by,

$$\chi = \lim_{V \rightarrow \infty} \frac{d_{AB}d_{CV}}{d_{AC}d_{BV}} = \frac{1}{\lambda} \lim_{V \rightarrow \infty} \left(1 + \frac{d_{CB}}{d_{BV}}\right) = \frac{1}{\lambda} \quad (15)$$

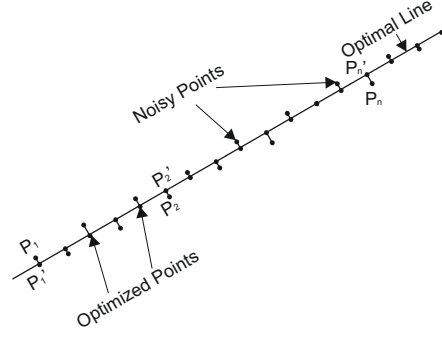


Figure 6: Noisy image points and its projection on best fit line

In the image frame, let the coordinates of V' be given by Eq. 13, We write the cross ratio as,

$$\chi' = \frac{\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}}{\sqrt{(u_1 - u_3)^2 + (v_1 - v_3)^2}} \times \frac{\sqrt{\left(\frac{u_3 - u_3\alpha - u_1 + u_2\alpha}{1 - \alpha}\right)^2 + \left(\frac{v_3 - v_3\alpha - v_1 + v_2\alpha}{1 - \alpha}\right)^2}}{\sqrt{\left(\frac{u_2 - u_1}{1 - \alpha}\right)^2 + \left(\frac{v_2 - v_1}{1 - \alpha}\right)^2}} \quad (16)$$

Substituting the value of α from Eq. 12 and simplifying algebraically, we get $\chi = \chi'$. This shows that the cross ratio of four points is invariant under projection and our VP estimates are accurate.

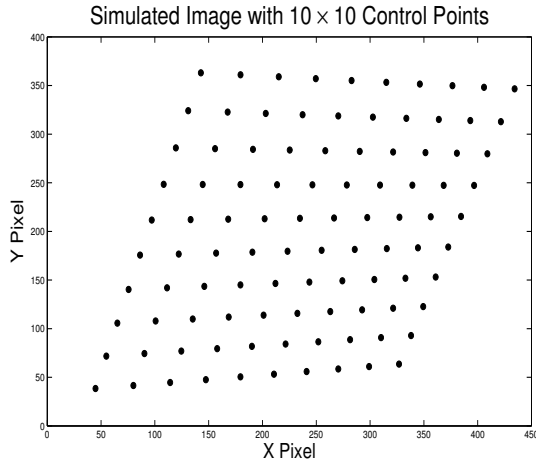
3.3 Tackling Noise

In the presence of noise, the performance of image processing techniques may get degraded. If the location of those three collinear points is not known precisely, errors will creep in to the VP estimates. To reduce this sensitivity we incorporate a least squares based optimization method. The idea behind this method is to draw a least square fit line from the selected three points to find the direction. Then orthogonally project these points on this line. These new points are used in place of earlier noisy data see Fig. 6.

4 RESULTS

Experiments were performed to validate the 3-Point (3-PVP) method. Robustness of the method and its computational efficiency are investigated through simulations. All simulations are performed in MATLAB® environment. Simulations were performed on a PC with i5 processor (3.2 GHz, 64 bit) and 4 Gb RAM.

[4] and [3] focus on vp estimation in urban/man-made environment where determining distance ratio will be difficult and will have to be separately estimated. Hence, we compare our results with LSVP method described in Section 2.

Figure 7: Simulated Pattern of 10×10 control points

4.1 Simulation

We here simulate perspectivity transform and generate synthetic images. A target pattern with 10×10 evenly distributed points is simulated as shown in Fig. 7. Two sets of parallel lines can be drawn in X and Y direction. This pattern is projected on the synthetic image-plane using a simulated camera. Properties are tabulated in Table 1.

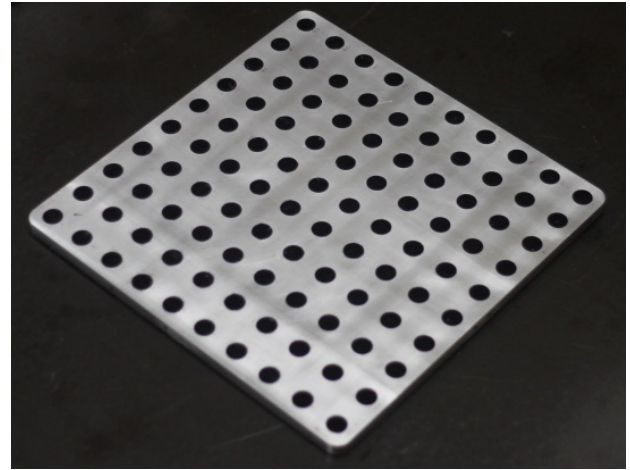
Absolute ground truth can be found out for simulated images and hence it can be a great tool to validate estimation method. Since, we are simulating perspective, the camera matrix \mathbb{P} is known to us and homogenous coordinates (of infinity point along the line) are known to be $[1, 0, 0, 0]$.

Synthetic images provide us with an unique opportunity to add Gaussian noise and analyse robustness. Gaussian noise $\mathcal{N}(0, 0.1 \text{ px})$ is added to each projected point on the pattern. This new noisy image is given as input to the VP estimation algorithm. Two vanishing points are estimated in each image, represented by VP1 and VP2. We perform 100 Monte-Carlo runs for both methods.

Accuracy and Robustness

We compare our results with least square approach. 3-PVP provides accurate VP estimates. This is seen from the fact that 3-PVP estimates are closer to the ground truth. The mean error and standard deviation of error are also lower for 3-PVP as compared to LSVP. The mean errors are negligibly small as compared to VP estimate for both methods. Standard deviation of error is approximately one third the value of LSVP. Results are tabulated in Table 4.

Noise with std of 0.1 px was used to study robustness. 3-PVP method is shown to be robust to image noise. Euclidean norm of error is much higher for LSVP as compared to 3-PVP. Error norm for LSVP is approximately three times that of 3-PVP. Error values are tabulated in Table 3

Figure 8: Metal fixture with 10×10 control points

Parameter	Value
Focal Length	50 mm
Principal Point	(360,240) px
Skew Factor	0
Scale Factor	1
Orientation Vector	$[0^0 \ 40^0 \ 30^0]$
Translation Vector	$[800, -1200, 300]$ mm
Image Resolution	720×480

Table 1: Simulated camera parameters

Fig. 9 and Fig. 10 shows the ground truth location of VP and the spread of estimated VPs using both methods. The VPs estimated by LSVP have more deviation from the true value. Fig. 11 represents the error in x and y direction along with error norm for the Monte-Carlo runs. 3-PVP method shows lesser error as compared to LSVP.

Computational Cost

Monte-Carlo runs also can indicate the computational cost of the algorithm. The time taken by both methods are tabulated in Table 2 for 100, 1000 and 10000 runs. We can observe that the speed of 3-PVP is approximately ten times faster. LSVP method involves firstly forming least square lines and secondly finding their intersection. 3PVP on the contrary employs an algebraic relation and is hence fast. These simulations validate our method's accuracy, robustness and speed.

Number of Runs	3-PVP (s)	LSVP (s)
100	0.230184	1.27458
1000	1.5083	13.09765
10000	14.9408	132.65978

Table 2: Simulation Time

4.2 Experimental Results

The target used for validating the 3-P VP method is shown in Fig. 8. The coordinates of the center of circles are known with high degree of accuracy. These

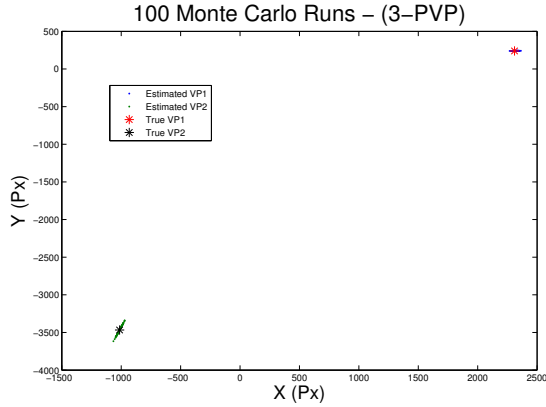


Figure 9: VP estimate spread for 3-PVP

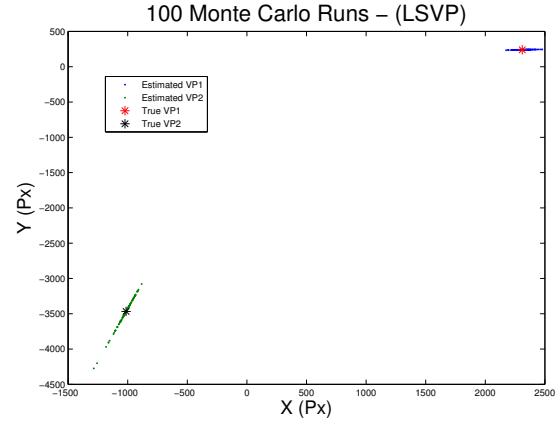


Figure 10: VP estimate spread for LSVP

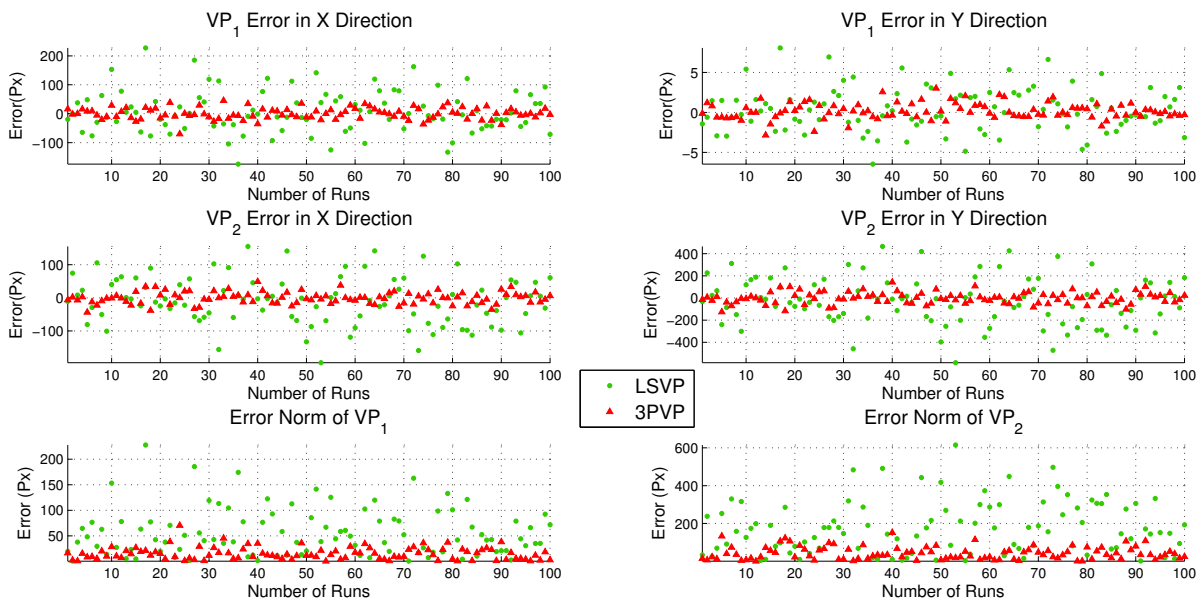


Figure 11: Error in VP Estimates (RMS, X & Y Direction) for 3-PVP v/s LSVP

points are planar in nature and form two sets of parallel lines. The image is captured using a Cannon EOS 1100 D camera with fixed effective focal length of 50 mm. Median filter has been used to remove noise from the image. A well-focused image of the target is processed in MATLAB to obtain centroids of the circular control points. Two vanishing points are obtained from each image. The vanishing points obtained by the 3-p strategy are compared with results from LSVP method.

3-PVP and LSVP are used on three images and their VPs are estimated. The results show that both the methods work effectively with the current image. The distance between results from both methods is shown to be of the order of 10^{-08} or lower. This also shows that in the absence of noise both methods will converge to the same estimate. VP estimates for those three images are tabulated in Table 5.

The advantages of our method are,

- *Tackling of radial distortion*: Our algorithm gives us the freedom to select the three points, which can be selected such that they lie in the middle of the image. Radial distortion effects are negligible near the center.
- *Handling defocused images*: Partial defocusing in images can lead to large errors in feature extraction. We can select required three points in such a way that you avoid defocused parts of the image.
- *Independent of Parallel Lines*: Errors also creep in when the given set of lines is not perfectly parallel. We do not need parallel lines and hence are not prone to errors.
- *Fast and Robust*

VP	True VP (px)	3-PVP Error Norm		LSVP Error Norm	
		Mean (px)	STD (px)	Mean (px)	STD (px)
VP ₁	(2310.1, 240)	15.59	11.81	50.36	38.09
VP ₂	(-1013.1, -3467.3)	52.37	37.75	166.2	146.8

Table 3: Comparison of mean and std of error norm in VP estimation using 3-PVP and LSVP

VP	True VP (px)	3-PVP Error		LSVP Error	
		Mean (px)	STD (px)	Mean (px)	STD (px)
VP _{1,x}	2310.1	15.5410	11.8248	50.3179	38.073
VP _{1,y}	240	0.6822	0.5821	1.9273	1.4071
VP _{2,x}	-1013.1	16.8240	12.1201	52.6534	46.5807
VP _{2,y}	-3467.3	49.5547	35.8124	157.5836	139.2167

Table 4: Comparison of 3-PVP and LSVP estimates and error analysis

Image	VP	3-PVP (px)	LSVP (px)	$\ \varepsilon\ (\text{distance})$
Image 1	VP ₁	(279.45, -1179.07)	(279.45, -1179.07)	6 E -11
	VP ₂	(-36027.08, 4453.91)	(-36027.08, 4453.91)	1 E -08
Image 2	VP ₁	(2601.13, -1539.28)	(2601.13, -1539.28)	2 E -10
	VP ₂	(-1778.97, -859.70)	(-1778.97, -859.70)	1 E -10
Image 3	VP ₁	(1301.67, -1121.90)	(1301.67, -1121.90)	1 E -10
	VP ₂	(-4198.81, -644.39)	(-4198.81, -644.39)	1 E -10

Table 5: Vanishing Points of real images using 3-PVP and LSVP

5 CONCLUSION

The 3-PVP method is based on single point perspective. Three collinear points and their distance ratio in the world frame characterize perspective in the direction of that line. It provides us with a non-iterative, closed-form solution. It is proved to be accurate and robust. VP estimation was performed on simulated images with a gaussian noise $\mathcal{N}(0,0.1)$. It provides VPs with approximately one third the error norm and a smaller standard deviation as compared to LSVP. 3-PVP method is shown to be computationally cheap. Experimental results show that in the absence of noise, 3-PVP method and LSVP converge to the same value (accurate estimation) albeit with much lesser computational time. It has promising future in applications which require high accuracy VP estimation.

ACKNOWLEDGMENT

We gratefully acknowledge the help and guidance provided by Mr. R S Chandrasekhar, RCI.

6 REFERENCES

- [1] Hartley, Richard and Zisserman, Andrew: Multiple View Geometry in computer vision, Cambridge University Press (2000)
- [2] Forsyth, D., Ponce A., Computer Vision: A Modern Approach, 2nd Edition: Prentice Hall, ch 16.1, pp. 437-439 (2011)
- [3] Tardif, J.P., Non-iterative Approach for Fast and Accurate Vanishing Point Detection, 12th IEEE International Conference on Computer Vision, Kyoto, Japan, pp. 1250-1257 (2009)
- [4] Barinova, O. et. al., Geometric Image Parsing in Man Made Environments. 11th European Conference on Computer Vision, pp. 57-70 (2010)
- [5] Denis, P., Elder, J. H., Estrada, F. J.: Efficient Edge-based Methods for Estimating Manhattan Frames in Urban Imagery. ECCV 2008, Part II, LNCS 5303, pp. 197-210 (2008)
- [6] Tsai, R.Y.: A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE J. Robotics Automat., Vol. RA-3, No. 4, pp. 323-344 (1987)
- [7] Grammatikopoulos, L., Karras, G., Petsa, E., Camera calibration combining images with two vanishing points. Int. Arch. of Photogrammetry, Remote Sensing and Spatial Information Sciences, 35 (Part 5), pp. 99-104 (2004)
- [8] Li, B. et. al.: Simultaneous vanishing point detection and camera calibration from single images. Proceedings of the 6th international conference on Advances in visual computing, pp. 151-160, Volume Part II (2010)
- [9] Caprile, B., Torre, V., Using vanishing points for camera calibration. Int. Journal of Computer Vision, 4(2), pp. 127-140 (1990)

- [10] Lee, D. H., Jang K. H., and Jung, S. K.: Intrinsic Camera Calibration Based On Radical Center Estimation. The 2004 International Conference on Imaging Science, Systems, and Technology, USA, pp. 7-13, (2004)
- [11] He. B.W., Li Y.F., Camera calibration from vanishing points in a vision system, Optics and Laser Technology, Volume 40, pp. 555-561(2008)
- [12] Orghidan, R. et al.: Camera calibration using two or three vanishingpoints, Proceedings of the Federated Conference onComputer Science and Information Systems, pp. 123-130 (2012)
- [13] Fong, C.K.: 3D object reconstruction from single distortedline drawing image using vanishing points. Proceedings of ISPACS 2005 pp. 53-56 (2005)
- [14] Parodi, P. and Piccioli, G.: 3D shape reconstructionby using vanishing points. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18,no. 2, pp. 211-217 (1996)
- [15] Schuster, R., Ansari, N. and Bani-Hashemi,A.: Steering a Robot with Vanishing Points. IEEE-Transactions on Robotics and Automation,Vol. 9, NO. 4, pp. 491-498 (1993)
- [16] Wang, Y.: An efficient algorithm for UAV indoorpose estimation using vanishing geometry. 12th IAPRConference on Machine Vision Applications, Japan, pp. 361-364 (2011)

MCM-CBIR: Multi Clustering Method for Content Based Image Retrieval

Hadjer LACHEHEB

LRIA Laboratory, Computer
Science Department, USTHB
Algiers, Algeria

hlacheheb@usthb.dz

Saliha AOUAT

LRIA Laboratory, Computer
Science Department, USTHB
Algiers, Algeria

saouat@usthb.dz

Izem HAMOUCHENE

LRIA Laboratory, Computer
Science Department, USTHB
Algiers, Algeria

ihamouchene@usthb.dz

ABSTRACT

Image retrieval systems are designed to provide the ability of searching and retrieving images in huge image databases. A content based image retrieval system (CBIR) is used to offer such tasks based on the content of the image. In this paper we propose a new method of CBIR system based on a learning technique. Our method uses k-means clustering to reduce data and to improve the system performance. The specificity of our approach is the use of each feature vector separately in the clustering process in order to obtain different clustering on the same database, differently to other approaches that combine features vectors to cluster the database. For this reason we call it multi-clustering approach. The advantage of this approach consists in keeping the performance of the features and getting several views of the database due to the separation of features. The experimental results show the efficiency of our approach.

Keywords

CBIR, visual features, color, texture, shape, k-means clustering.

1. INTRODUCTION

The increasing number of digital images makes the information management hardest. CBIR is the process of retrieving images from a huge database on the basis of visual features. These features are automatically extracted, such as color, texture and shape. The greater parts of CBIR systems are based on a typical architecture shown (see Fig.1). This architecture includes two phases. The first phase is the offline phase or Data insertion [Tor]. This phase consists of the extraction of features vectors or descriptors of each image in the database. The second phase is the online or Query processing [Tor]. In this phase we extract the query feature vectors and compare it to all feature vectors of the database. We compare the query image to the other images in the database with computing a similarity measure between their feature vectors. Afterwards, the obtained images will be ordered following a decreasing order of similarity. The visual contents

or Features are the representation of any distinguishable characteristic of an image [Sub02].

These features require three levels: low, middle and high. Low level features are the visual content that can be extracted from the information obtained in the pixel level such as color, texture, and shape.

One of the attractive parts in an image is the color. The color space is the identification of the color; it is generally represented using three elements. For instance, RGB (red, green, blue) color space, HSV (hue, saturation, value), CIE $L^*a^*b^*$ [Col04].

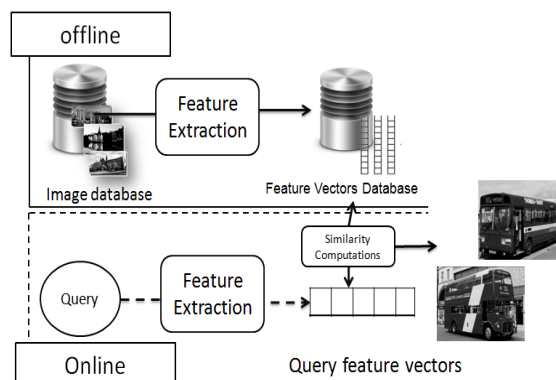


Figure 1. Typical architecture of CBIR systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

The texture is a powerful feature that is neglected generally by a lot of CBIR systems. But, until now a clear definition about texture does not exist. However, we can define it as the repeated model that has the proprieties of homogeneity, coarseness, contrast, directionality, line-likeness, regularity, roughness [Zha12]. The other important descriptor or feature is a shape.

Shape is the characteristic surface represented by a contour or an outline. Shape descriptors need a good segmentation into regions or objects. Zhang and Lu [Zha04] classify shapes feature methods for boundary based and region based. All these descriptors need similarity metrics for the purpose of comparing the query image features vectors and feature vectors of the images in the database. A similarity measure determines the distance between the feature vectors (low level features) representing the images. Some of famous measurements are Euclidean distance, Minkowski distance, Manhattan distance, and histogram intersection [Swa91]. Many researchers use other methodologies or tools of artificial intelligence like machine learning. Arthur Samuel [Sam59] defines the machine learning as a field that gives computers the ability of learning without being programmed. Two major categories are introduced in machine learning which are supervised and unsupervised learning. In supervised learning a supervisor is going to teach the computer on the other side in the unsupervised learning the computer learns by itself. Clustering is a famous technique of unsupervised learning. One of the commonly used clustering techniques is K-means clustering. K-means is used to find K different clusters in a database of N objects, where similarities between objects in the same cluster are minimized, and between objects of the other clusters are maximized [Tan05], [Jai11].

This paper is organized as follows:

Section 2 is an overview about related works dealing with CBIR. The third section contains the details about our proposed method. In section 4 we show the experimental results and comments about their efficiency after applying our proposed technique.

2. RELATED WORKS

Many general-purpose image retrieval systems have been developed. A lot of famous systems like QBIC [Fal94], Photobook [Pen96], Blobworld [Car02], Virage [Gup97], VisualSEEK and WebSEEK [Smi96]. An important part of new approaches start to use key point features one of the well known proposition on that are SIFT descriptors proposed by Lowe in 2004 [Low04]. Another used approach is visual words where an image is represented by a

histogram of visual words [Fei05],[Siv03]. Besides, using clustering techniques is an efficient and an important option added to CBIR systems since they allow reducing the time of retrieval and increasing the performance of research. K-means was early proposed over 50 years ago it is still one of the most widely used algorithms for clustering. The main reasons for the success of these techniques are the ease of implementation, the simplicity, the efficiency and the empirical success. The most efficient k-means algorithm is EIKAN's algorithm [Jai11]. For instance, SemQuery system [She02] organizes images into different groups of clusters based on their heterogeneous features. Vailaya et al. [Vai01] create a hierarchical structure about vacation images. At the top level, images are classified as indoor or outdoor. Outdoor images are then classified as a city or landscape that are further divided into the sunset, forest, and mountain closes. The SIMPLiCity system [Wan01] category images into a graph, textured photograph, or non-textured photograph. In 2012, Swapna Borde and Udhav Bohosle proposed novel techniques for image retrieval using clustering features extracted from images which are RMC, CMC, RMDC and CMDC, RMWC and CMWC. Other techniques use the advantages of transforms (wavelet and DCT) [Rai12]. Other useful MPEG-7 for searching in multimedia systems are in [Sal02,Bas10, Say05]. For instance, VITALAS (Video & image Indexing and reTrievAl in the LARge Scale) is an Industrial project started from 2007 the aim of this project is to produce a prototype for industry to retrieve and index multimedia information. This project has given an interesting result. The goal of this project focus on: First, Cross-media indexing and retrieving try to use automatic annotation and getting semantic level. Second, using techniques for large scale search. Finally, trying to improve visualization and context adaptation[Vit07].

3. THE PROPOSED METHOD

Our approach uses k-means clustering on each feature of the images separately. For each feature vector we get a different clustering. For example, if we extract color feature vector and texture feature vector and shape feature vector we get color clustering, texture clustering and shape clustering. These differences clustering allow us to have different views and levels of the retrieved images for a query which is not possible with other approaches. We call this method Multi-Clustering Content Based

Image Retrieval. The proposed approach takes four steps:

The two first steps are in the offline phase of our CBIR system and the two second steps are in the online phase.

1. Feature extraction.
2. K-means multi-clustering.
3. Query image searching.
4. Organizing results.

3.1. Feature Extraction

The first step is to extract features from the images of the database. V_i is a feature vector. We choose color and texture features for our system. Images used are in RGB color space. In addition, we propose to represent color feature with three feature vector: red histogram, green histogram and blue histogram. Also, we use the gray level co-occurrence matrix GLCM to represent texture feature.

3.1.1 Histogram

A color histogram [Swa91] is an important feature we can extract gray level histogram as we can use color histogram. For our system, we extract histograms with 256 bins of each R, G, B colors. So, as a result we get three feature vectors for color descriptors.

3.1.2 Co-occurrence Matrix (GLCM).

The Grey Level Co-occurrence Matrix, or called the Grey Tone Spatial Dependency Matrix) is a table representing a number of different combinations of pixel brightness values (grey levels) that occur in an image [Har73].

At the end of this step four feature vectors are extracted

- V1: histogram of red color.
- V2: histogram of green color.
- V3: histogram of blue color.
- V4: co-occurrence matrix.

To clarify more this first step we illustrate an example (see Fig.2). To simplify this example we suppose that we have three feature vectors for each image in the database

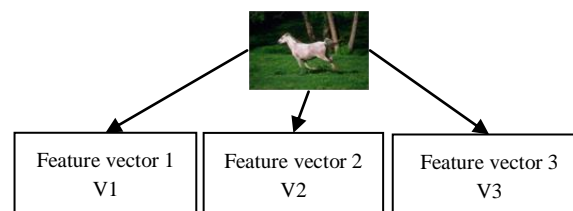


Figure 2. Illustration of the feature extraction for one image.

3.2. K-Means Multi-Clustering

After the feature extraction, k-means clustering algorithm is executed. K-means clustering is a learning machine algorithm. This algorithm needs 80% of images for the learning phase and 20% of images for the evaluation phase.

The evaluation images are the images used to query the system. Learning k-means clustering is to find K different clusters in a database of N objects, where similarity intra cluster distance is minimized, and the distance inter clusters are maximized [Tan05], [Jail1].

The learning machine is repeated until the centers of the clusters are stable. Algorithm 1 shows the operation of k-means clustering [Tan05].

Algorithm k_means_clustering

Begin

Select K points as the initial centroid.

Repeat

Form K clusters by assigning all points to the closest centroid.

Recompute the centroid of each cluster.

Until the centroids don't change.

End.

Algorithm 1: k-means clustering [Tan05]

At the end of this step we obtain k clusters with K stable centers and each N element assigned to its corresponding cluster. This learning is executed for each previous vectors V_i , $i=1 \dots 4$. For our example we have just V_1 , V_2 and V_3 (see Fig.3).

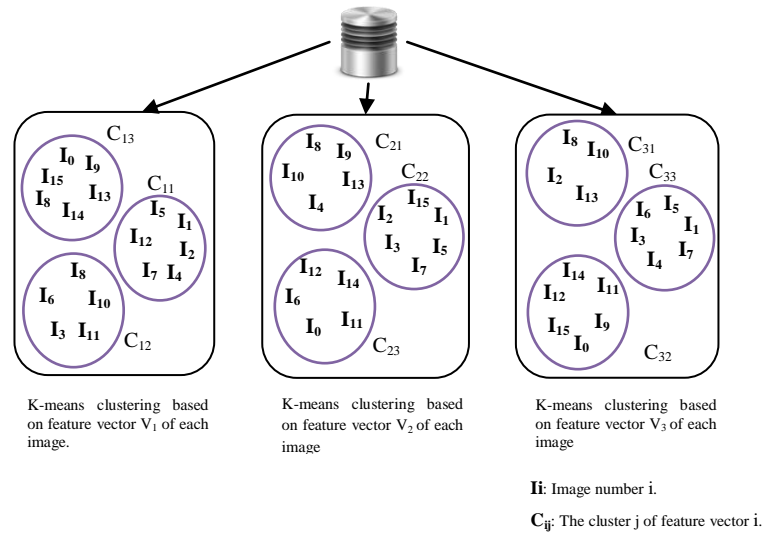


Figure 3. Example of a multi-clustering data base (offline phase)

3.3. Query Image Searching

The two previous steps are in the offline phase. Next two steps are in the online phase. First, we extract the feature vectors of the query image using the same process (see Fig.2). As a result we get four feature vectors (VQ_1, VQ_2, VQ_3, VQ_4).

For each feature vector, a searching process is launched. Each query feature vector VQ_i is searched in its corresponding clustering. For instance, VQ_1 is searched in V_1 clustering. As a result we get the relevant cluster of each feature vector VQ_i (see Fig.4).

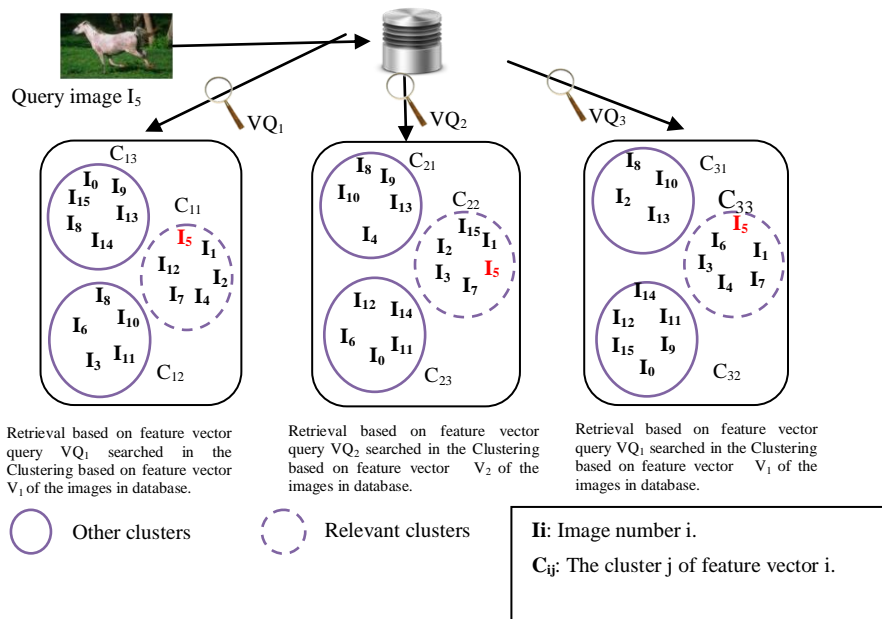


Figure 4. Query processing (online phase)

3.4. Organizing Results

The last step is to organize the results from the most to the less relevant. We propose two ways in this step. First, organization by levels. Second, organization by the clusters union. Let us carry on the previous example, we take I_5 as image query. As a result we have three relevant clusters (C_{11} , C_{22} , C_{33}) (see Fig.4).

3.1.3 First Proposed Organization (By Levels).

In this organization we propose to separate the results on three levels. In the first level we display the intersection of all relevant clusters. In our case $C_{11} \cap C_{22} \cap C_{33} = \{I_5, I_1, I_7\}$. For the second level. The intersection two by two of relevant clusters that do not exist in the previous level. For our example: $((C_{11} \cap C_{22}) \cup (C_{22} \cap C_{33}) \cup (C_{11} \cap C_{33})) - (C_{11} \cap C_{22} \cap C_{33}) = \{I_2, I_4, I_3\}$. Third level. The images that exist in just one cluster and do not exist in the previous two levels. For our example $C_{11} \cup C_{22} \cup C_{33} - ((C_{11} \cap C_{22}) \cup (C_{22} \cap C_{33}) \cup (C_{11} \cap C_{33})) - (C_{11} \cap C_{22} \cap C_{33}) = \{I_{12}, I_{15}, I_6\}$.

After, for each level we compute the similarity measure (Euclidian distance) and order the retrieved images in a descending order.

3.1.4 Second Proposed Organization (By a Clusters Union).

The second proposed organization is to group all images in just one level and compute the similarity measure and order the retrieved images in a descending order. In our example, $C_{11} \cup C_{22} \cup C_{33} = \{I_1, I_2, I_4, I_7, I_{12}, I_{15}, I_3, I_6\}$.

4. RESULTS AND DISCUSSION

This section is a presentation of our experimental results. In addition, we are going to compare our results to other clustering CBIR system methods like WaveQ [Geb07] and discuss these results. To experiment our approach, we use the Corel database [Wan01]. The database contains 1000 images divided on 10 classes as follows: African people and villages, Beach, Buildings, Buses, Dinosaurs, Elephants, Flowers, Horses, Mountains and Glaciers, Food. We choose to extract color and texture features. To process our method we first divide our dataset into two parts, where 80% of the images are used during the training data set (offline phase) and 20% of the images are used for the query phase (online phase). During the offline phase, we pre-process the images, extract their feature vectors and construct a new database containing features vectors of each image. After, we apply k-means clustering for each feature vector. In the online phase, we calculate the distance between the query image and the images in the database. Finally, we sort these distances and return

all the found relevant images after using our two organization methods presented in section 3. Moreover, a presentation of the results without using texture features (only color features) and then we add other results using both features (color and texture). We evaluate our system with computing recall and precision [Per01].

$$\text{Recall} = \frac{\text{number of relevant images retrieved}}{\text{number of relevant images in collection}} \quad (1)$$

$$\text{Precision} = \frac{\text{number of relevant images retrieval}}{\text{total number of images retrieved}} \quad (2)$$

4.1. Without Texture Feature

We execute our system on several images without using texture features. So we use just three feature vectors (red, green, blue). The results are impressive and in most case relevant 20 images are displayed on the top of the retrieved images.

We test for ($K=5$) or five clusters. From 69 tested images average recall is above 0.77 and average precision is over 0.33. In addition, we get 94% of querying images having a precision above 0.50. Also, we get 52% of images having more than 0.90 recall value. We notice that for five classes we get better results.

4.2. With Texture Feature

In this part we add the co-occurrence matrix to the three other features. The results are good and promising. We test for ($K=5$) and we get the following results the average of recall is above 0.918. Comparing to the previous tests without texture, texture features increase the recall. This means that the number of retrieved relevant images increases.

4.3. Comparison With Other Systems

We will compare our system with WaveQ system. WaveQ uses a clustering method for this database. An execution is shown (see Fig.5) where we can notice the efficiency of our system comparing to WaveQ. WaveQ gives as results the image query at first and then a dish in the second position. Also, the other images are semantically the same buses. Our system displays the image query as first image and the four other images are buses. In addition, our results are visually the same (red and white buses color) and semantically the same (bus in the city).



Figure 5. An example comparing WaveQ system and our system. Set of Images number 1 are results of WaveQ system. Sets numbers two are the five first retried images of our system without texture. Sets numbers three are the five first images results of our system with texture.

We can see another test of our system. First, A multi level organization is displayed (see Fig.6). In addition, we extract images intersection of the relevant clusters images (see Fig.7).

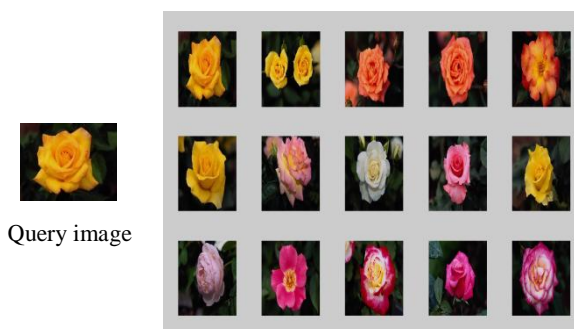


Figure 6. An example of levels organization results in our system. First five relevant images are displayed for each level

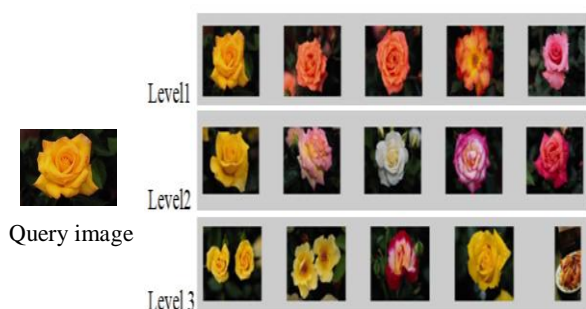


Figure 7. An example of second proposed organization method of our system. Twenty first relevant image are displayed

5. CONCLUSION AND FUTURE WORK

In this paper a new approach for content based image retrieval is proposed. This method uses K-means

clustering separately for each feature vector of an image. This is done in order to keep the efficiency of each feature without combining them to get one feature vector. Also, the proposed approach avoids the computing of similarity measures for the entire database, we just calculate those of the relevant clusters. The results show the effectiveness of our approach and give a good Recall and a promising precision values in database of 1000 images. In addition, our system gives very satisfactory retrieved images (20 first images look alike the query). Comparing to WaveQ our system gives visually better results.

As future work, we will test this method with different color space, color features and texture features. In addition, using others similarity measures.

References

- [Bas10] Bastan, M. , Cam, H., Gudukbay, U. and Ulusoy, O. BilVideo-7: An MPEG-7-Compatible Video Indexing and Retrieval System,” *IEEE Multimedia*, vol. 17, no. 3, pp. 62–73, 2010.
- [Car02] Carson, C., Belongie, S., Greenspan, H. and Malik, J. Blobworld: image segmentation using expectation-maximization and its application to image querying. *Pattern Analysis and Machine Intelligence*, IEEE Transactions on. Vol. 24, pp.1026-1038, 2002.
- [Col04] The Color Guide and Glossary, Communication, measurement and control for Digital Imaging and Graphic Arts X-rite, 2004.
- [Fal94] Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D. and Equitz, W. Efficient and effective Querying by Image Content. *Journal of Intelligent Information Systems*. Vol. 3, pp.231–262 , 1994.
- [Fei05] Fei-Fei, L. and Perona, P. A Bayesian hierarchical model for learning natural scene categories. *Computer Vision and Pattern Recognition*, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 2, pp. 524 – 531, 2005.
- [Geb07] Gebara, D. and Alhaji, R. WaveQ: Combining Wavelet Analysis and Clustering for Effective Image Retrieval. *Advanced Information Networking and Applications Workshops*, 2007, AINAW '07. 21st International Conference on. pp.289–294, 2007.
- [Gup97] Gupta, A. and Jain, R. Visual information retrieval. *Commun. ACM*. 40, 70–79, 1997.

- [Har73] Haralick, R.M., Shanmugam and K., Dinstein, I.: Textural Features for Image Classification. Systems, Man and Cybernetics, IEEE Transactions on. SMC-3, pp.610–621, 1973.
- [Jai11] Jain, M., Singh and S.K. A Survey On: Content Based Image Retrieval Systems Using Clustering Techniques for Large Data sets. International Journal of Managing Information Technology. Vol. 3, pp.23–39, 2011.
- [Low04] Lowe, D. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. Vol. 60, pp.91–110, 2004.
- [Pen96] Pentland, A., Picard, R.W. and Sclaroff, S. Photobook: Content-based manipulation of image databases. International Journal of Computer Vision. Vol. 18, pp.233–254, 1996.
- [Per01] Müller, H. , Müller, W., Squire, S. Marchand-Maillet, D. M. and Pun, T. performance evaluation in content based image Retrieval: overview and proposals, Pattern Recognition Letters, vol. 22, no. 5, pp. 593–601, April 2001 .].
- [Rai12] Raikwar, A.K. and Jain, S. Article: Content based Image Retrieval using Clustering. International Journal of Computer Applications. 41, pp.29–33, 2012.
- [Sal02] Salembier, P. and Sikora, T. Introduction to MPEG-7: Multimedia Content Description Interface. New York, NY, USA: John Wiley & Sons, Inc., 2002.
- [Sam59] Samuel, A.L. Some studies in machine learning using the game of checkers. IBM J. Res. Dev. Vol. 3, pp.210–229, 1959.
- [Say05] Saykol, E., Gündükbay, U. and Ulusoy, Ö. A histogram-based approach for object-based query-by-shape-and-color in image and video databases, *Image Vision Comput.*, vol. 23, no. 13, pp. 1170–1180, 2005.
- [She02] Sheikholeslami, G., Chang, W., and Zhang, A. SemQuery: semantic clustering and querying on heterogeneous features for visual data. Knowledge and Data Engineering, IEEE Transactions on. Vol. 14, pp.988–1002, 2002.
- [Siv03] Sivic, J. and Zisserman, A. Video Google: a text retrieval approach to object matching in videos. Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on. Vol.2, pp. 1470–1477, 2003.
- [Smi96] Smith, J.R. and Chang, S.-F. VisualSEEK: a fully automated content-based image query system. Proceedings of the fourth ACM international conference on Multimedia. pp. 87–98. ACM, New York, NY, USA, 1996.
- [Sub02] Subramanya, S.R., Teng, J.-C. and Fu, Y. Study of Relative Effectiveness of Features in Content-Based Image Retrievals. Proceedings of the First International Symposium on Cyber Worlds (CW'02). IEEE Computer Society, Washington, DC, USA, pp.0168–175, 2002.
- [Swa91] Swain, M.J. and Ballard, D.H. Color indexing. International Journal of Computer Vision. Vol. 7, 11–32, 1991.
- [Tan05] Tan, P.-N., Steinbach, M. and Kumar, V. Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [Tor] Torres, R.D.S., Falcão, A.X. Content-Based Image Retrieval: Theory and Applications. Revista de Informática Teórica e Aplicada. Vol. 13, pp.161–185.
- [Vit07] VITALAS (Video & image Indexing and reTrievAl in the LARge Scale) <http://vitalas.ercim.eu/>.
- [Vai01] Vailaya, A., Figueiredo, M.A.T., Jain, A.K. and Zhang, H.-J. Image classification for content-based indexing. Image Processing, IEEE Transactions on. Vol.10, pp. 117–130, 2001.
- [Wan01] Wang, J.Z., Li, J. and Wiederhold, G. SIMPLiCity: semantics-sensitive integrated matching for picture libraries. Pattern Analysis and Machine Intelligence, IEEE Transactions on. vol. 23, pp.947–963, 2001.
- [Zha04] Zhang, D. and Lu, G. Review of shape representation and description techniques. Pattern Recognition. Vol. 37, pp.1–19, 2004.
- [Zha12] Zhang, D., Islam, M.M. and Lu, G. A review on automatic image annotation techniques. Pattern Recognition. Vol. 45, pp. 346–362, 2012.

Generic Fitted Primitives (GFP): Towards Full Object Volumetric Reconstruction for Service Robotics

Tiberiu T. Cocias
Institute of Automation
Transilvania University of
Brasov
tiberiu.cocias@unitbv.ro

Florin Moldoveanu
Institute of Automation
Transilvania University of
Brasov
moldof@unitbv.ro

Sorin M. Grigorescu
Institute of Automation
Transilvania University of
Brasov
s.grigorescu@unitbv.ro

Abstract

Service robotics applications, such as mobile manipulation in domestic environments, require 3D representations of the objects of interest to be grasped. Simple object recognition or segmentation cannot provide structural shape information mandatory for obtaining reliable grasp configurations. In this paper, the *Generic Fitted Primitives* (GFP) technique for volumetric reconstruction is introduced. The goal of the method is to build full 3D object shapes from a single camera perspective. In order to obtain the shape of the 3D primitive, we propose an energy-minimization algorithm based on the concept of *Active Contours* applied directly on 3D visual data. Our modeling approach produces compact closed surfaces (*volumes*) describing the objects of interest which can be further used for service robotics tasks, such as grasping or manipulation. The performance of the proposed technique has been evaluated against two different methods, i.e. generalized active contours and superquadric approximations.

Keywords

Active contours, 3D segmentation, 3D reconstruction, Robot vision systems, RGB-D sensors.

1 INTRODUCTION

In the last years, the 3D object reconstruction challenge gained a lot of attention in application fields such as service and industrial robotics, or virtual reality. In service robotics, 3D reconstruction is usually involved in providing information for path planning and object grasping in mobile manipulation [Dil09a]. In such cases, one major inconvenience regarding a service robot is that it can perceive the scene from only one camera perspective. This aspect produces large occluded areas altering the structure of the objects. Thus, an estimation of the object's 3D shape must be considered in order to obtain a full volumetric representation. Some methods try to reconstruct directly the volume of the object by discretizing the 3D space into a series of voxels [Bet00a]. Other approaches aim at defining implicit surfaces depicting different volumes through implicit functions [Bar02a]. In this paper, we propose the *Generic Fitted Primitives* (GFP) technique for fully approximating the particular volumetric information of the objects. The calculated models are intended to be used for improving the grasping capabilities of service

robots. The input visual information has been acquired using structured light sensors, such as the MS Kinect®, and classical stereo cameras. An example of full 3D reconstructed objects from a typical service robotics scene is illustrated in Fig. 1.

In our GFP approach, the problem of 3D volumetric approximation has been divided into two phases. Firstly, a coarse object *detection* method is used to extract an initial object cluster for which its volume needs to be estimated. In the second phase, the cluster is used for fitting a GFP such that detected object will be fully reconstructed. The main contributions of the paper may be summarized as follows:

- the introduction of the GFP technique based on a modified formulation of the *Active Contours* principle; the deformation of the primitive shape is performed based on the normal direction of the so-called *control points* of the GFP;
- the usage of a GFP as an initial contour within the active contours framework; the modeling process time is thus improved since the number of iterations required to deform the initial shape is smaller;
- usage of the GFP approach for building full 3D volumetric models of objects of interest in the context of mobile manipulation.

3D object surface reconstruction is treated in a large number of publications. Some of the paper found in lit-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.



Figure 1: Full 3D volumetric reconstruction of multiple objects in a mobile manipulation scenario.

erature are based on the implicit representation of models [Bae02a, Bar02a], whereas other exploit explicit a priori surfaces (e.g. skeleton primitives) for augmenting the existing object structure [Gas01a]. One relevant implicit approach makes use of generic 3D shapes (e.g. such as spheres, cuboid, or ellipses) to roughly approximate the global object volume [Bar81a]. These types of methods are fast, do not need any a priori knowledge about the reconstructed surface, but lack the accuracy of the final approximated volume. A more refined volume can be obtained by partitioning the imaged surface in more meaningful sub-regions which can be further individually approximated using the same implicit principle as in [Coc12a].

A different approach to 3D modeling is based on *3D Object Retrieval* (3DOR) search engines [Tan08a]. The main drawback of this technique is that it needs a high amount of computational power to find the optimal match between a query representation (e.g. the imaged object) and a set of targets (predefined models from a database). In [Hua12a], the combination of RANSAC and Procrustes analysis is used for recovering the joint axes of objects. The algorithm does not make use of any a priori object knowledge, but it requires a large number of images depicting the object of interest. In [Mar09a], the authors present a primitive based approach for approximating simple regulated objects like plates, boxes, cans, etc. As opposed to our work, in [Mar09a], the primitives are represented by simple geometric models, such as cuboids, spheres, or cylinders, and not by primitive shapes that can capture different particularities that the objects might have. Krainin et al. [Kra11a, Kra11b], applied the concept of object tracking during manipulation for building online 3D models of objects using range sensors and 3D data processing techniques. Nevertheless, the model is represented by a *Point Distribution Model* (PDM) and not by a full 3D shape.

The rest of the paper is organised as follows. In Section 2 the components of the primitive based modeling apparatus, along with the involved methodology, is

presented. Performance evaluation results are given in Section 3, before the conclusions from Section 4.

2 METHODOLOGY

In mobile manipulation, activities of daily living scenarios typically involve a large numbers of objects. The first step in the proposed framework is to segment the different objects and obstacles in the scene. As a result of segmentation, different 3D object clusters, or PDMs, are obtained. Along with the clustering procedure, the process also returns the object's class. These PDMs are used for modeling the GFP in such a way that it captures the particularities of the object. The block diagram of the GFP architecture is shown in Fig. 2.

Instead of using a large number of models as a priori information about a particular object, thus requiring a large number of shapes to be stored, we propose a more general approach through the use of GFPs. By using only one primitive per object class (e.g. mug, plate, bottle etc.), a considerably smaller sized primitives database is obtained. At the same time, the computation time is improved because the number of discrete items that need to be searched is reduced.

2.1 Initialisation: Cluster Extraction

The extraction of the scene clusters is important for the accuracy of the final primitive representation. Firstly, the objects of interest need to be recognised in order to select the correct GFP. We use a contextual object recognition approach [Son11a] through a classification process. Having obtained the object's class, the corresponding 3D cluster of the same object is extracted. The clusters are extracted as described in [Rus09a]. Namely, plane segmentation is used to divide an organized point cloud P into smaller meaningful clusters $C = [c_0, c_1, \dots, c_n]$ representing different entities. In Fig. 3 an example of object recognition (labelling) and object cluster extraction for a table-top scene is presented.

The output of the detection component is an isolated PDM representation of the objects of interest. Further, this representation is used to model the shape primitive such that in the end it models the particularities of the object as accurate as possible.

2.2 Generic Primitives

A generic primitive is considered to be an a priori known shape describing a number of particular objects from the same class. It is constructed in such a manner that it resembles many similar objects. In this way, an universal model for a certain class of objects is obtained. For example, different types of bottles can be roughly approximated by a joint pattern. The most important attribute of a shape is actually its global structure (*frame*) which, in a majority of cases, is similar to a

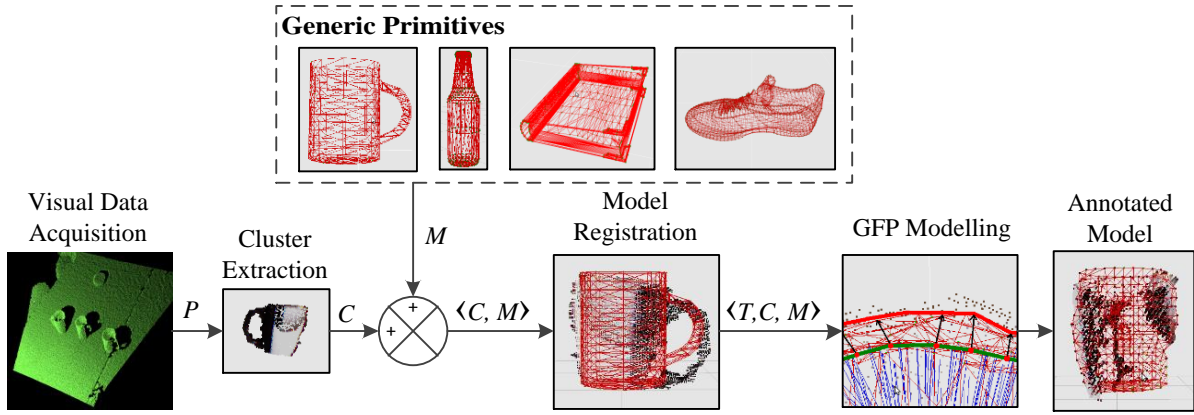


Figure 2: Block diagram of the GFP 3D volumetric estimation approach.

large number of objects of the same class. In this sense, instead of finding the optimal object (from a considerably large number of different shapes from the same class) which best fits the PDM data, a modeling step applied to a generic primitive M is addressed for estimating its global object volume.

Depending on the geometric surface complexity, the generic primitive can be defined by a high density of 3D points. This aspect directly influences the processing time. A down-sampling filter used to reduce the number of PDM points is not encouraged because the global point cloud structure is altered. We approach this issue from the GFP's point of view. Namely, not all feature points are relevant for modeling the cluster's structure. For example, many of them are used only for the purpose of creating a volumetric surface. In this sense, each point in the GFP will receive a special flag or type. Hence, two point types are defined: *control* and *regular* points. A point which has received the *control* flag is considered to be part of the frame and it is positioned according to the 3D information in the point cloud, whereas a *regular* point is used simply to smooth the global structure of the shape, meaning it is moved according to the positions of the control points.

The classification of GFP points in different types can be done either manually or automatically. The first procedure requires a human to manually select the point's type. Having in mind the required human interaction, the manual labelling of points is time consuming. On the other hand, the *automatic* type assignment has a lower accuracy, but it is usually much more efficient and does not suffer from subjectivity.

The automatic point selection is governed by a set of rules used to establish the point's type [Cot95a]. Namely, control points are those obeying the following statements:

- points describing sharp corners of a boundary, detected as in [Web10];
- points marking the boundaries of M along the widest axis;
- points located at equal distance around a boundary between two control points obeying rule one;
- points marking a curvature extreme or the extreme points of the object [Wat01].

An example GFP of a bottle is illustrated in Fig. 4.

In terms of the GFP definition, the primitive model is a complex data structure composed of a vector M storing the 3D coordinates of all the features describing the generic model, a vector A containing the point type attributes of M , a vector W describing the mesh triangulation indexes used for 3D surface representation and, in the end, global characteristics such as height, length, width, rotation R , translation T and the overall number of features.

The objects are defined in a *local* coordinate system attached to each considered model. Thus, a *common* coordinate frame for both the GFP models and the cluster of the object of interest, must be computed.

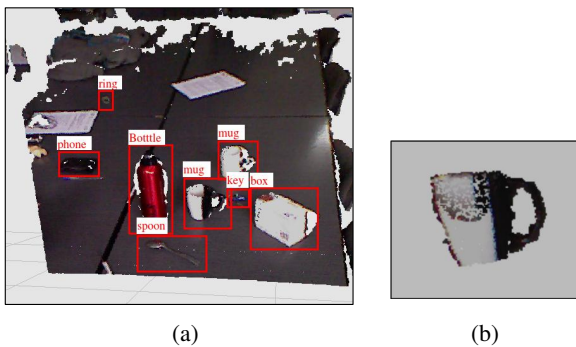


Figure 3: Object detection through cluster extraction. (a) Scene labelling based on object recognition. (b) Euclidean cluster extraction.

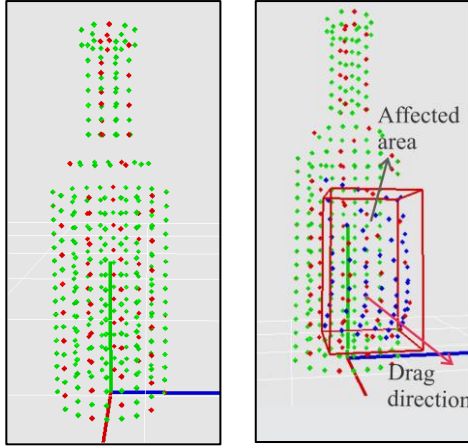


Figure 4: Generic primitive of a bottle. Control points are marked with red, while regular points are green. Modelled points are shown in blue.

2.3 Model Registration

In order to correctly transfer the particularities of the considered object to the GFP model, the involved shapes must be aligned. For this purpose, the scene model frame is considered to be the reference frame. The primitive will be initially aligned according using a rigid body transformation. Therefore, the rotation R , translation T and scale s of the GFP has to be determined relative to the scene.

The scale factor s between the GFP and the segmented cluster is determined by approximating each cluster using a circumscribed sphere. The ratio between the radii of the shapes will act as a scale factor which will resize the primitive to the size of the object.

By subtracting the center of mass $m_M(x, y, z)$ of the primitive M from the center of mass $m_C(x, y, z)$ of the object cluster c_i , a relative translation $T_{3 \times 1}$ can be obtained. Finally, the rotation $R_{3 \times 3}$ is determined by incrementally rotating the primitive along all the three axes and minimizing a sum of Euclidean distances between the closest corresponding neighbor points of the two forms. Further, a fine model fitting is obtained through the *Iterative Closest Point* (ICP) algorithm [Zan94]. In Fig. 5(a), 5(b) and 5(c) the registration of a mug primitive is depicted.

Having computed all the prerequisite information for the final modeling process, the primitive cloud is aligned to the scene object using as:

$$M_{new}(i) = s \cdot R(M_{old}(i) + T), \quad i = 0 \dots size(M), \quad (1)$$

where $M_{new}(i)$ are the new coordinates of the primitive point and $M_{old}(i)$ are the initial point coordinates.

2.4 Primitive modeling

The purpose of the modeling process is to release the primitive model from his generality. Through this step,

the primitive will capture the *local* geometry information directly from the scene. Since initially no reliable information regarding the global structure can be identified, the modeling procedure occurs at a local level around each primitive point. If a particular vicinity lacks sensed information, the GFP will fill up the missing information with generic data, that is, the stored volumetric information in its shape. To make the entire process time efficient, the modeling process will occur only for *control* points while the regular points will be repositioned relative to these *control* points using a linear motion law. The basic principle underlying the primitive modeling step is known as *Active Contours* or *Snakes* [Kas1988]. In the initial formulation, a snake is a 2D curve which moves through an image domain driven by a set of energies computed based on particular image features. The behavior of a snake in the 3D space can be approximated with the weaving of a textile material. A major drawback of a snake is his inflexibility to topological changes. To cope with this issue, in [McI00], *topological snakes* (*T-Snakes*) are presented. Using an affine cell decomposition [All03], the authors succeeded to create in this sense a framework that significantly extends the abilities of standard snake model.

The initial snake representation is described by a small circle in the 2D image domain [Kas1988], while its analogous in 3D is a sphere. Instead of using a sphere as the starting closed surface, we address the usage of a generic primitive, which already stores a rough structure of the considered object [Coc12b]. In comparison with [Coc12b], in this paper the movement of a contour point is constrained to only two directions, given by the normal direction. Thus, an important computations reduction is achieved.

In 3D, a *snake* structure is harder to control because of the extra degrees of freedom introduced by the third dimension. While for the 2D case there are only 8 possible moving directions, for 3D the number of candidate directions reaches 26 (given by the grid representation of the space). The *Active Contours* method tries to minimize a functional of energies $\mathcal{E}(c)$ in order to incrementally sculpt the initial contour c to an optimal final form as described in Eq. 2. Two types of energies are formulated for this purpose. The first type, E_{int} , is used to constrain the model deformation such that the structure integrity of the shape is kept at any moment during the modeling process, while the second one, E_{ext} , drives the considered modelled point to a its best candidate position:

$$\min \left(\sum_0^N (E_{int} + E_{ext}) \right), \quad (2)$$

and

$$E_{int} = \alpha(i) \cdot E_{cont}(i) + \beta(i) \cdot E_{curv}(i), \quad (3)$$

where N is the number of modelled feature points, α and β are the internal energy weight factors, E_{cont} represents the energy which ensures that the model surface is continuous (such that during the modeling process the newly rearranged points will not produce large gaps) and E_{curv} is the energy responsible for the smoothness of the surface. The α and β parameters are constrained by an empirical established threshold value. Thus, if the respective energy has a value below that particular threshold, then the respective weight factor (α and β), will be set 0, otherwise it will be 1.

As in Eq. 3, the internal energy is composed of two energies: E_{cont} and E_{curv} . They are used exclusively to constrain the movement of the points and at the same time to keep the model as compact and intuitive as possible. The internal energies are computed based on the first and second derivatives of the points which are to be modelled. The computation of the derivative of a certain snake point implies a neighborhood knowledge of the contour points. For example, in 2D the first derivative of a snake point is computed based on the previous and current position of the considered point. Similar, the second derivative is computed using the position of the previous, current and next point in the snake contour. In the 3D space, the previous respective the next snake contour points are evaluated as the closest, respective second closest nearest neighbor. Nevertheless, this approach is not always correct. At sharp corners this type of selection is erroneous. To avoid that, the relations between the points of the countour should be established using the mesh like representation. In this approach, the points making up the 3D contour will be the vertices of a mesh. Each face of the mesh describe a relation between minimum 3 points, thus the correct previous, respectively next countour points can be easily established. Considering these aspects, the mathematical formulation of the internal energy computed in a certain contour point p_i , can be stated as follows:

$$\begin{aligned} E_{cont}(i) &= \left| \frac{dc}{ds} \right|^2 + \left| \frac{dc}{dr} \right|^2 \\ &= \|p_i(s) - p_{i-1}(s)\|^2 + \|p_i(r) - p_{i-1}(r)\|^2, \end{aligned} \quad (4)$$

respectively,

$$\begin{aligned} E_{curv}(i) &= \left| \frac{d^2c}{ds^2} \right|^2 + \left| \frac{d^2c}{dr^2} \right|^2 + \left| \frac{d^2c}{dsdr} \right|^2 \\ &= \|p_{i-1}(s) - 2p_i(s) + p_{i+1}(s)\|^2 + \\ &\quad \|p_{i-1}(r) - 2p_i(r) + p_{i+1}(r)\|^2 + \\ &\quad \|p_{i-1}(s, r) - 2p_i(s, r) + p_{i+1}(s, r)\|^2 \end{aligned} \quad (5)$$

where, s and r are the axis used to represent the 3D contour (as an topological manifold)

The process of minimizing the functional of energy $\mathcal{E}(c)$ implies resolving the following Euler-Lagrange equation:

$$\begin{aligned} E_{ext} + \alpha(s) \left| \frac{dc}{ds} \right|^2 + \alpha(r) \left| \frac{dc}{dr} \right|^2 - \beta(s) \left| \frac{d^2c}{ds^2} \right|^2 - \\ \beta(r) \left| \frac{d^2c}{dr^2} \right|^2 - \beta(s, r) \left| \frac{d^2c}{dsdr} \right|^2 = 0. \end{aligned} \quad (6)$$

The equation is true when the energies used in the process are in equilibrium. This also means that the contour has touched a relevant characteristic from the space (corner, edge, etc.).

The most important energy in our context, E_{ext} , is represented, in the 2D domain by the intensity of the grey-scale candidate pixel. Similarly, in 3D, the pixel's intensity is equivalent to the neighboring density of points. The amount of neighbors lying in a given area is determined using the *kdtree* principle [Ben75a]. To avoid searching for the optimal candidate trough all 26 possible directions of a control point, given by the grid based representation of the space, the *normal* direction is used to reduce the search space to only 2 candidate directions. Fig. 5(d) shows the computed normal of a given generic primitive. Moving a control point exclusively along his normal directions deforms the surface in a natural and intuitively manner. By doing this, the overall processing time is considerably improved. Along the normal direction, the best position candidate for the control point is determined using the next set of rules:

- if the primitive candidate point m_{cd} is already lying in a dense region, move m_{cd} along both normal directions and find the first point with the number of nearest neighbors nn_{cp} closest to 0: $nn_{cp} \geq 0$;
- if m_{cd} has $nn_{cp} = 0$, search along both normal direction for $nn_{cp} > 0$; if, after searching, $nn_{cp} = 0$, freeze the control point in its initial position since no reliable surface information was found; if $nn_{cp} \neq 0$, set the position of the control point in the candidate position (with $nn_{cp} > 0$) closest to m_{cd} .

Based on this set of rules, the best candidate position of a given modelled point can be determined. In Fig. 5(e), the movement of a control point towards the local density information is illustrated. The final model of the GFP is depicted in Fig. 5(f).

To help create a more smoother surface and reduce the number of snake iterations, we propose an Euclidean distance based linear constrained. When a point is

moved from an initial to a candidate position, all the neighbors within a vicinity radius equal to the Euclidean distance between the initial and the candidate position, will be gradually affected by a linear factor as:

$$M_{new}(i) = M_{old}(i) \cdot \left(1 + \frac{d_{curr}}{d_{max}}\right), \quad i = 0 \dots size(M) \quad (7)$$

where $M_{new}(i)$ and $M_{old}(i)$ are the new and old coordinates of the points lying inside the affected area of radius d_{max} . d_{curr} is the Euclidean distance between the current affected point and the control point. d_{max} is the Euclidean distance between the farthest point inside the affected area. Figs. 4 and 5(e) show the behavior of such Euclidean linear constraint principle, where a single point is dragged along the normal direction to find the correct surface location.

Inside the sphere described by the d_{max} radius, the points are modified accordingly to a computed ratio based on the distance between the neighboring and the initial primitive points. In this sense, the deformation is gradually applied, having the greatest effect on the neighbor points lying closer to the considered control point. Thus, the neighbor points at the margin of the sphere are slightly deformed. The proposed algorithm, not only is time efficient, but the movement of the points occur in a more intuitively way.

3 EXPERIMENTAL RESULTS

3.1 Setup

For evaluation purposes, two types of sensors have been used: a MS Kinect[®] RGB-D sensing device, used mainly for indoor testing, and a Point Grey Bumblebee[®] stereo camera for acquiring outdoor scenes. Both sensors output dense 3D information from the imaged scene. During tests, a constant illumination was ensured.

As GFP models, we have used the benchmark database in [Shi04a]. By using GFPs, the size of the database has been reduced from 1814 objects to only 142 general primitives. The GFPs were created as average shapes of the initial database. The point type assignment of all the primitives in the database was performed offline using the automated process, which took, in average, around 8 minutes for each model. During testing, all objects were placed on flat surfaces for detection and segmentation¹.

¹ The source code of the GFP approach is part of the ROVIS machine vision system, available at the svn repository <http://rovis.unithbv.ro/rovis/>. Please ask the authors permission for downloading.

3.2 Metrics

For evaluation purposes, an Euclidean based fitting measurement has been considered. Because the main goal of the modeling approach is to create a particular representation of a GFP, the distance between these two representations can be considered to be a similarity measure. Thus, by summing the distances between each point from the scene's objects and the nearest neighboring GFP point, the following fitting metric is obtained:

$$fit_{dist}(C, M) = \frac{1}{N} \sum_{i=0}^n \frac{1}{1 + \underset{fit_{dist}}{\operatorname{argmin}} ||C(i) - nn_i(M)||^2 \cdot \gamma}, \quad (8)$$

where $fit_{dist} \in [0, 1]$ is the fitting error and N the number of points in the GFP. C and M are the PDMs of the clustered object and of the modeled GFP, respectively. $C(i)$ is the closest scene point to a GFP point $nn_i(M)$, while γ represents a scale factor. The better the GFP modeling is, the lower the value of fit_{dist} is.

3.3 Case Study: modeling a Mug

From a total number of 410 primitive points describing a mug, only 203 (107 *control* points and 96 *regular* points) were moved during the modeling process. The rest of the points were assessed as optimal positions since they do not have any 3D data in their vicinity. The modeling computation time is approx. 680ms. The final mug model was obtained after only 21 modeling iterations. At each step, the contour was pushed through the scene with a 1mm offset further along the normal directions. Concerning the original formulation with the initial contour depicting a sphere, the number of iterations needed to obtain a shape similar to the modelled primitive is around 38 iterations. A comparative analysis of the energy evolution is illustrated in Fig. 6. The computation time, for the case of the sphere, reached 9sec.

For a different mug shape (e.g. highly deformed cup), the automatic point type assignment will generate a greater number of control points, directly influencing the computation time. Having a larger number of points describing the structure, the computation time increased from 0.6sec. to 0.9sec. for the GFP modeling.

For objects describing complex surfaces, keeping the primitive defined through a low number of 3D points will cause the binding of some irregularities on the object surface. An up-sampling filter can be used to augment the initial primitive representation [Bre05a] and to produce high accurate models. When using denser representations, the fitting metric fit_{dist} is lower than the one obtained from sparse representations. Particular to the

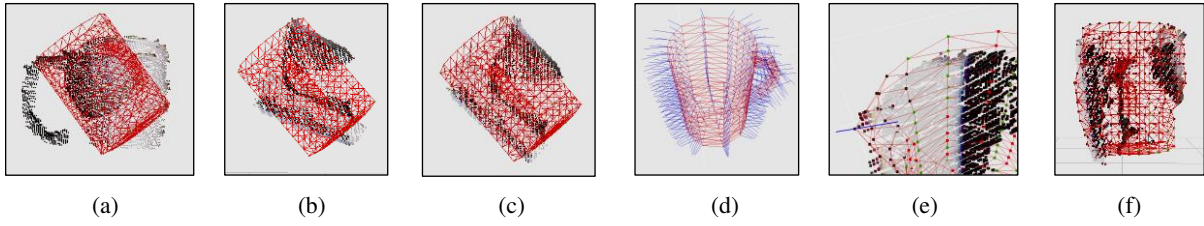


Figure 5: GFP modeling of a mug. (a) Initial overlapping between the object's cluster and the GFP. (b) Euclidean distance based rotation approximation. (c) Fine alignment using ICP. (d) Primitive points normals described by the straight blue lines. (e) Searching along the normal directions. (f) Final annotated GFP.

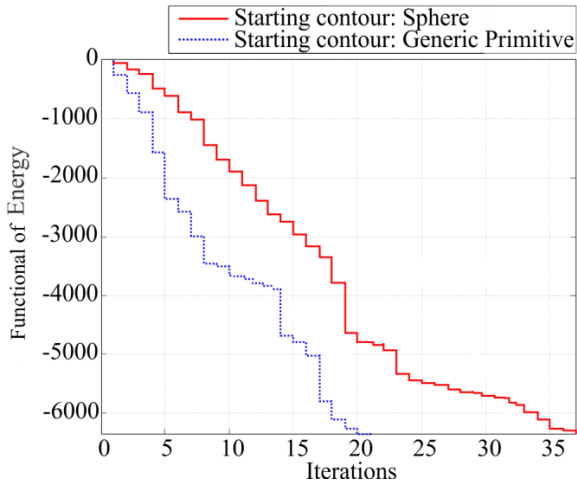


Figure 6: Number of iterations required for modeling using for different initial contours. A comparative analysis between the GFP approach and 3D active contours.

mug model, the overall volumetric error has been lowered to 5% by using the up-sampled representation of the same shape.

3.4 GFP vs. Superquadrics

Among the existing object volumetric estimation methods, *superquadrics* represent a real competitor for the GFP principle presented in this paper. A *superquadric* is a parametrized geometric shape obtained by the spherical product between two curves modelled through a series of parameters [Bar81a]. It resembles many geometrical models starting from simple cubes or cylinders and ending with complex ones such as toroid or hyperboloid. Because of the roughness provided by the generic shape, it is not desirable to use only one superquadric to approximate an object. In this sense, multiple joined superquadrics produces a more precise and fine object [Coc12a].

By constantly changing the 11 parameters which defines a superquadric and by evaluating the newly obtained shape through an *in-out* function, the optimal model, given the point cloud representation of a particular object, can be obtained. This principle is considered to be fast because only a few parameters, which actually

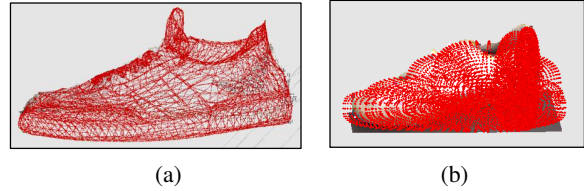


Figure 7: Estimated object volume described as a red point cloud. (a) GFP technique. (b) Multiple superquadrics approach.

deform the output shape, are controlled. The complex structure of a particular object can be approximated, in some case, with a very large number of superquadrics. Indeed, the global object volume will be more precise if this value is large, while the computation time will exponentially increased. On the other hand, by using a small number of superquadric models, only a rough volume will be obtained in the end. It can be stated that for simple objects, the superquadric based method is faster, whereas for complex models, the GFP technique excels both on precision, as well as on time efficiency.

One major advantage of superquadrics, in contrast to the GFP technique, is that it does not need pose normalization or any a priori knowledge regarding the class of the segmented object. This is the compromise that the GFP method is paying for its precision. In Fig. 7 a shoe modeling example using both methods is presented. Comparative numerical results are given in Table 1.

Method	Processing time [sec]	Fitting accuracy [%]
Superquadrics	4.79	0.7689
GFP	1.53	0.9762

Table 1: Comparative results between Superquadric based volume estimation and the GFP technique for the case of a shoe.

By using a primitive as an initial contour, the volumetric fit error of the GFP method is the smallest. Using superquadrics, the final volume is obtained as a reunion of a series of superellipses which best approximate the segmented object. Since the superquadric approach doesn't need pose normalization, the volumetric fit error is the only comparison parameter used during

the comparative evaluation. From the grasping point of view, both methods output reliable grasp models, the difference being that the GFP technique, because of its accuracy, provides more grasping configurations. The grasping configurations were calculated using the GraspIt [Mil01a] methodology. A grasping simulation for a shoe object is depicted in Fig. 8.

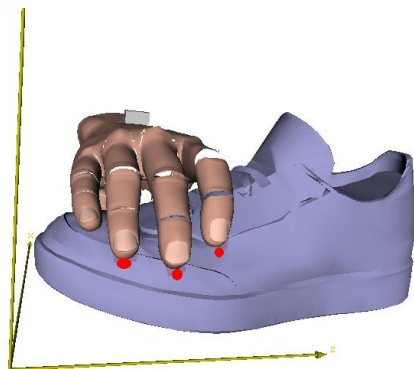


Figure 8: Grasp candidate configuration for a shoe. The world frame axes are coloured in yellow whereas the pressure points intersecting the GFP are marked with red.

4 CONCLUSIONS

In this paper, the GFP 3D object volumetric estimation technique has been presented. The goal of the approach is to estimate as accurately as possible the 3D structure of objects found in robotic mobile manipulation scenarios. As future work the authors consider the time computation enhancement of the proposed procedure through parallel computational devices (e.g. Graphic Processors), as well as the application of the method to other computer vision related areas, such as 3D medical imaging.

ACKNOWLEDGMENT

This paper is supported by the Sectoral Operational Program Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the projects POS-DRU/107/1.5/S/76945 and POSDRU/89/1.5/S/59323.

5 REFERENCES

- [All03] Allgower, E.L. and Georg, K. Introduction to Numerical Continuation Methods, in Classics in Applied Mathematics, Philadelphia, USA: SIAM, 2003
- [Bae02a] Baerentzen, J.A. and Christensen N.J. Volume sculpting using the level-set method, in Proceedings of the Shape Modeling, USA, 2002, pp.175-182.
- [Bar81a] Barr A.H. Superquadrics and angle-preserving transformations, IEEE Computer Graphics and Applications, 1, pp.11-23, 1981.
- [Bar02a] Museth, K. and Breen, D.E. and Whitaker, R.T. and Barr, A.H. Level set surface editing operators, in ACM Transactions on Graphics, 2002, pp.330-338.
- [Ben75a] Bentley, J.L. Multidimensional binary search trees used for associative searching, in Commun. ACM, 18, No.9, pp.509-517, Sep. 1975.
- [Bet00a] Betz, A. 3-D object reconstruction using spatially extended voxels and multi-hypothesis voxel coloring, in Proceedings of the Intern. Conf. on Pattern Recognition, USA, 2000, pp.1774-1782.
- [Bre05a] Breitenkopf, P. and Naceur, H. and Rassineux, A. and Villon, P. Moving least squares response surface approximation: Formulation and metal forming applications, in Computers and Amp Structures, 83, No.18, pp.1411-1428, 2005.
- [Coc12a] Cocias, T.T. and Grigorescu, S.M. and Moldoveanu, F. Multiple-superquadrics based object surface estimation for grasping in service robotics, in 13th Intern. Conf. on Optimization of Electrical and Electronic Equipment, 2012, pp.1471-1477.
- [Coc12b] Cocias, T.T. and Grigorescu, S.M. and Moldoveanu, F. Object volumetric estimation based on generic fitted primitives for service robotics, in VISAPP (2), 2012, pp.191-197.
- [Cot95a] Cootes, T.F. and Taylor, C.J. and Cooper, D. and Graham, J. Active shape models-their training and application, in Comp. Vision and Image Understanding, 61, No.1, pp.38-59, 1995.
- [Dil09a] Huebner, K. and Welke, K. and Przybylski, M. and Vahrenkamp, N. and Asfour, T. and Kragic, D. and Dillmann R. Grasping known objects with humanoid robots: A box-based approach, in Intern. Conf. on Advanced Robotics, 2009, pp.1-6.
- [Gas01a] Ferley, E. and Cani, M.P. and Gascuel, J.D. Resolution adaptive volume sculpting, in Graph. Models, 63, No.6, pp.459-478, Nov. 2001.
- [Hua12a] Huang, X. and Walker, I.D. and Birchfield, S. Occlusion-aware reconstruction and manipulation of 3D articulated objects, in Intern. Conf. on Robotics and Automation, 2012, 1365-1371.
- [Kas1988] Kass, M. and Witkin, A. and Terzopoulos, D. Snakes: Active contour models, in Intern. Journal on Computer Vision, 1, No.4, pp.321-331, 1988.
- [Kra11a] Krainin, M. and Henry, P. and Ren, X. and Fox, D. Manipulator and object tracking for in-hand 3D object modeling, in Intern. Journal of Robotic Research, 30, pp.1311-1327, Sep. 2011.
- [Kra11b] Krainin, J. and Curless, B. and Fox, D. Autonomous generation of complete 3D object mod-

- els using next best view manipulation planning, in Proceedings of the IEEE Intern. Conf on Robotics and Automation, Shanghai, China, May 2011.
- [Mar09a] Marton, Z.C. and Goron, L.C. and Rusu, R.B. and Beetz, M. Reconstruction and verification of 3D object models for grasping, in Intern. Symp. on Robotics Research, 2009, pp.315-328.
- [McI00] McInerney, T. and Terzopoulos, D. T-Snakes: Topology adaptive snakes, in Medical Image Analysis, 4, no. 2, pp. 73-91, 2000.
- [Mil01a] Miller A. GraspIt!: A Versatile simulator for robotic grasping, Columbia University, 2001.
- [Rus09a] Rusu, R.B. Semantic 3D object maps for everyday manipulation in human living environments, Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, 2009.
- [Shi04a] Shilane, P. and Min, P. and Kazhdan, M. and T. Funkhouser, The princeton shape benchmark, in Shape Modeling International, 2004.
- [Son11a] Song, Z. and Chen, Q. and Huang, Z. and Hua, Y. and Yan, S. Contextualizing object detection and classification, in Proceedings of the 2011 IEEE Conf. on Computer Vision and Pattern Recognition, USA, 2011, pp.1585-1592.
- [Tan08a] Tangelder J.W. and Velkamp R.C. A survey of content based 3D shape retrieval methods, in Multimedia Tools Appl., 39, No.3, pp.441-471, Sep. 2008.
- [Zan94] Zhang, Z. Iterative point matching for registration of free-form curves and surfaces, in Intern. Journal on Computer Vision, 13, No.2, pp.119-152, Oct. 1994.
- [Web10] Weber, C. and Hahmann, S. and Hagen, H. Sharp Feature detection in point clouds, in Proceedings of the 2010 Shape Modeling International Conference (SMI '10). IEEE Computer Society, Washington, DC, USA, 2010, pp. 175-186.
- [Wat01] Watanabe, K. and Belyaev, A. G. Detection of salient curvature features on Polygonal Surfaces, in Computer Graphics Forum, 20, No.3, pp. 385-392, 2001.

Straight Skeleton for Automatic Generation of 3-D Building Models with General Shaped Roofs

Kenichi Sugihara

Gifu Keizai University

5-50 Kitagata-chou

Ogaki city, Gifu-Pref., 503-8550, Japan

sugihara@gifu-keizai.ac.jp

ABSTRACT

3D urban models are important in several fields, such as urban planning and gaming industries. However, enormous time and labor has to be consumed to create these 3D models, using a 3D modeling software such as 3ds Max or SketchUp. In order to automate laborious steps, a GIS and CG integrated system is proposed for automatically generating 3D building models, based on building polygons (building footprints) on digital maps. Digital maps shows most building polygons' edges meet at right angles (orthogonal polygon). In the digital map, however, not all building polygons are orthogonal. In either orthogonal or non-orthogonal polygons, the new system is proposed for automatically generating 3D building models with general shaped roofs by straight skeleton computation. In this paper, the algorithm for shrinking a polygon and forming a straight skeleton are clarified and, the new methodology is proposed for constructing roof models by assuming 'the third event' and, at the end of the shrinking process, the shrinking polygon is converged to 'a line of convergence'.

Keywords

3D urban model, automatic generation, GIS (Geographic Information System), 3D building model, straight skeleton.

1. INTRODUCTION

3D urban models, such as the one shown in Fig.1 right, are important in urban planning and gaming industries. Urban planners may draw the maps for sustainable development. 3D urban models based on these maps are quite effective in understanding what if this alternative plan is realized. However, enormous time and labour has to be consumed to create these 3D models, using 3D modeling software such as 3ds Max or SketchUp. For example, when manually modeling a house with roofs by Constructive Solid Geometry (CSG), one must use the following laborious steps:

(1) Generation of primitives of appropriate size, such as box, prism or polyhedron that will form parts of a house (2) Boolean operations are applied to these primitives to form the shapes of parts of a house such as making holes in a building body for doors and

windows (3) Rotation of parts of a house (4) Positioning of parts of a house (5) Texture mapping onto these parts.

In order to automate these laborious steps, a GIS (Geographic Information System) and CG integrated system was proposed for automatically generating 3D building models, based on building polygons or building footprints on a digital map shown in Fig. 1 left [Sug09]. A complicated orthogonal polygon can be partitioned into a set of rectangles. The proposed integrated system partitions orthogonal building polygons into a set of rectangles and places rectangular roofs and box-shaped building bodies on these rectangles. In order to partition an orthogonal polygon, a useful polygon expression (RL expression: edges' Right & Left turns expression) and a partitioning scheme was proposed for deciding from which vertex a dividing line (DL) is drawn [Sug12].

In the digital map, however, not all building polygons are orthogonal. In either orthogonal or non-orthogonal polygons, the new system is proposed for automatically generating 3D building models with general shaped roofs by the straight skeleton defined by a continuous shrinking process, which was introduced and discussed by Aichholzer et al.[Aic95]. However, some roof models are not created by their

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

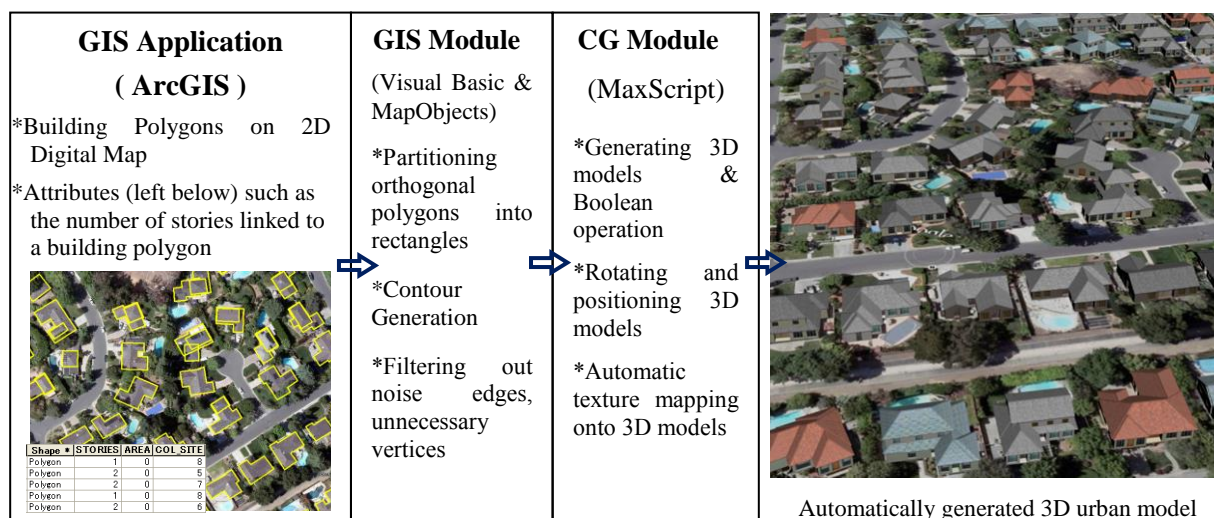


Figure 1. Pipeline of Automatic Generation for 3D Building Models

proposed method. In this paper, the new methodology is proposed for constructing roof models by assuming ‘the third event’ and, at the end of the shrinking process, the shrinking polygon is converged to ‘a line of convergence’.

2. RELATED WORK

Since 3D urban models are important information infrastructure that can be utilized in several fields, the researches on creations of 3D urban models are in full swing. Various types of technologies, ranging from computer vision, computer graphics, photogrammetry, and remote sensing, have been proposed and developed for creating 3D urban models.

Using photogrammetry, Gruen et al. [Gru98, Gru02] introduced a semi-automated topology generator for 3D building models: CC-Modeler. Feature identification and measurement with aerial stereo images is implemented in manual mode. During feature measurement, measured 3D points belonging to a single object should be coded into two different types according to their functionality and structure: boundary points and interior points. After these manual operations, the faces are defined and the related points are determined. Then the CC-Modeler fits the faces jointly to the given measurements in order to form a 3D building model.

Suveg and Vosselman [Suv02] presented a knowledge-based system for automatic 3D building reconstruction from aerial images. The reconstruction process starts with the partitioning of a building into simple building parts based on the building polygon provided by 2D GIS map. If the building polygon is not a rectangle, then it can be divided into rectangles. A polygon can have multiple partitioning schemes. To avoid a blind search for optimal partitioning schemes, the minimum description length principle is used. This principle provides a means of giving

higher priority to the partitioning schemes with a smaller number of rectangles. Among these schemes, optimal partitioning is ‘manually’ selected. Then, the building primitives of CSG representation are placed on the rectangles partitioned.

These proposals and systems, using photogrammetry, will provide us with a primitive 3D building model with accurate height, length and width, but without details such as windows, eaves or doors. The research on 3D reconstruction is concentrated on reconstructing the rough shape of the buildings, neglecting details on the façades such as windows, etc [Zla02].

On the other hand, there are some application areas such as urban planning and game industries where the immediate creation and modification of many detailed building models is requested to present the alternative 3D urban models. Procedural modeling is an effective technique to create 3D models from sets of rules such as L-systems, fractals, and generative modeling language [Par01].

Müller et al. [Mül06] have created an archaeological site of Pompeii and a suburbia model of Beverly Hills by using a shape grammar that provides a computational approach to the generation of designs. They import data from a GIS database and try to classify imported mass models as basic shapes in their shape vocabulary. If this is not possible, they use a general extruded footprint together with a general roof obtained by the straight skeleton computation defined by a continuous shrinking process [Aic95]. However, there is no digital map description and in-depth explanation about how the skeleton is formed and applied to create roofs in their paper.

More recently, image-based capturing and rendering techniques, together with procedural modeling approaches, have been developed that allow

buildings to be quickly generated and rendered realistically at interactive rates. Bekins et al. [Dan05] exploit building features taken from real-world capture scenes. Their interactive system subdivides and groups the features into feature regions that can be rearranged to texture a new model in the style of the original. The redundancy found in architecture is used to derive procedural rules describing the organization of the original building, which can then be used to automate the subdivision and texturing of a new building. This redundancy can also be used to automatically fill occluded and poorly sampled areas of the image set.

Aliaga et al. [Dan07] extend the technique to inverse procedural modeling of buildings and they describe how to use an extracted repertoire of building grammars to facilitate the visualization and modification of architectural structures. They present an interactive system that enables both creating new buildings in the style of others and modifying existing buildings in a quick manner.

Vanega et al. [Car10] interactively reconstruct 3D building models with the grammar for representing changes in building geometry that approximately follow the Manhattan-world (MW) assumption which states there is a predominance of three mutually orthogonal directions in the scene. They say automatic approaches using laser-scans or LIDAR data, combined with aerial imagery or ground-level images, suffering from one or all of low-resolution sampling, robustness, and missing surfaces. One way to improve quality or automation is to incorporate assumptions about the buildings such as MW assumption. However, there are lots of buildings that have cylindrical or general curved surfaces, based on non-orthogonal building polygons.

By these interactive modeling, 3D building models with plausible detailed façade can be achieved. However, the limitation of these modeling is the large amount of user interaction involved [Nia09]. When creating 3D urban models for urban planning or facilitating public involvement, 3D urban models should cover lots of citizens' and stakeholders' buildings involved. This means that it will take an enormous time and labour to model a 3D urban model with hundreds or thousands of building.

Thus, the GIS and CG integrated system that automatically generates 3D urban models immediately is proposed, and the generated 3D building models that constitute 3D urban models are approximate geometric 3D building models that citizens and stakeholder can recognize as their future residence or real-world buildings.

3. PIPELINE OF AUTOMATIC GENERATION

As the pipeline of automatic generation is shown in Fig.1, the source of a 3D urban model is a digital residential map that contains building polygons. The digital maps are stored and administrated by GIS application (ArcGIS, ESRI Inc.). The maps are then preprocessed at the GIS module, and the CG module finally generates the 3D urban model.

To streamline the building generation process, the knowledge-based system was proposed for generating 3D model by linking the building polygons to information from domain specific knowledge in GIS maps: attributes data such as the number of storey and the type of roof.

Preprocessing at the GIS module includes the procedures as follows: (1) Filter out an unnecessary vertex whose internal angle is almost 180 degrees. (2) Partition or separate orthogonal building polygons into sets of rectangles. (3) Generate inside contours by straight skeleton computation for placing doors, windows, fences and shop façades which are setback from the original building polygon. (4) Form the straight skeleton for the general shaped roof. (5) Rectify the shape of the polygons so that there are no gaps or overlaps between geometric primitives such as rectangles. (6) Export the coordinates of polygons' vertices, the length, width and height of the partitioned rectangle, and attributes of buildings.

The attributes of buildings, shown in Fig.1 left below, consist of the number of storeys, the image code of roof, wall and the type of roof (flat, gable roof, hipped roof, oblong gable roof, gambrel roof, mansard roof, temple roof and so forth). The GIS module has been developed using 2D GIS software components (MapObjects, ESRI).

As shown in Fig.1, the CG module receives the pre-processed data that the GIS module exports, generating 3D building models. In GIS module, the system measures the length and gradient of the edges of the partitioned rectangle. The CG module generates a box of the length and width, measured in GIS module.

In case of modeling a building with roofs, the CG module follows these steps: (1) Generate primitives of appropriate size, such as boxes, prisms or polyhedra that will form the various parts of the house. (2) Boolean operations applied to these primitives to form the shapes of parts of the house, for examples, making holes in a building body for doors and windows, making trapezoidal roof boards for a hipped roof and a temple roof. (3) Rotate parts of the house according to the gradient of the

partitioned rectangle. (4) Place parts of the house. (5) Texture mapping onto these parts according to the attribute received. (6) Copy the 2nd floor to form the 3rd floor or more in case of building higher than 3 stories.

CG module has been developed using Maxscript that controls 3D CG software (3ds MAX, Autodesk Inc).

4. STRAIGHT SKELETON COMPUTATION FOR ROOF GENERATION

Aichholzer et al. [Aic95] introduced the straight skeleton defined as the union of the pieces of angular bisectors traced out by polygon vertices during a continuous shrinking process in which edges of the polygon move inward, parallel to themselves at a constant speed. The straight skeleton is unexpectedly applied to constructing general shaped roofs based on any simple building polygon, regardless of their being rectilinear or not.

As shrinking process shown in Fig.2, each vertex of the polygon moves along the angular bisector of its incident edges. This situation continues until the boundary change topologically. According to Aichholzer et al. [Aic95], there are two possible

types of changes:

(1) **Edge event:** An edge shrinks to zero, making its neighboring edges adjacent now.

(2) **Split event:** An edge is split, i.e., a reflex vertex runs into this edge, thus splitting the whole polygon. New adjacencies occur between the split edge and each of the two edges incident to the reflex vertex.

A reflex vertex is a vertex whose internal angle is greater than 180 degrees.

All edge lengths of the polygon do not always decrease during the shrinking process. Some edge lengths of a concave polygon will increase. For example, as shown by 'ed1' and 'ed2' in Fig.2(a), the edges incident to a reflex vertex will grow in length. If the sum of the internal angles of two vertices incident to an edge is more than 360 degrees, then the length of the edge increases, otherwise the edge will be shrunk to a point (node).

Shrinking procedure is uniquely determined by the distance d_{shri} between the two edges of before & after shrinking procedure. The distance e_d_{shri} is the d_{shri} when an edge event happens in the shrinking process. e_d_{shri} for the edge (ed_i) is calculated as follows:

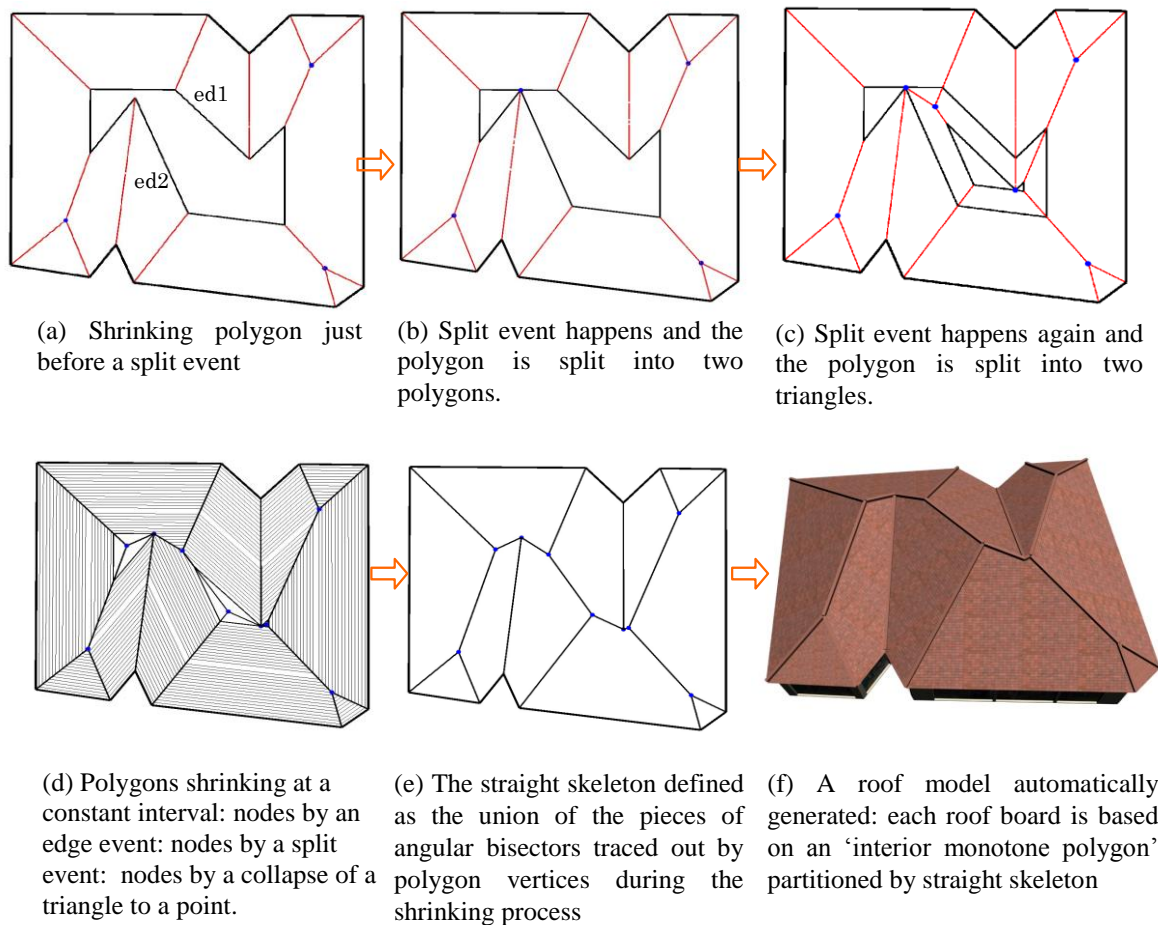


Figure 2. Shrinking process and a straight skeleton, a roof model automatically generated

$$e_d_{shri} = L_i / (\cot(0.5 * \theta_i) + \cot(0.5 * \theta_{i+1}))$$

where L_i is the length of ed_i , and θ_i & θ_{i+1} are internal angles of vertices incident to ed_i .

When $0.5 * \theta_i + 0.5 * \theta_{i+1} < 180$ degrees, i.e., the sum of the internal angles of two vertices incident to an edge is less than 360 degrees, an edge event may happen unless the edge is intersected by an angular bisector from a reflex vertex and a split event happens.

Fig.2 from (a) to (c) show a shrinking process for a non-orthogonal concave polygon: the polygon just before a split event: the polygon being split into two polygons after a split event happens. Fig.2(d) shows a set of polygons shrinking at the constant interval and nodes by an edge event and a split event, and nodes by a collapse of a triangle into a point.

Fig.2(e) shows the straight skeleton defined as the pieces of angular bisectors traced out by polygon vertices. Fig.2(f) shows the roof model automatically generated. Since the straight skeleton partitions the interior of a polygon with n vertices (n -gon) into n monotone polygons, each roof board that constitutes the roof model is formed based on these partitioned 'interior monotone polygons'.

4.1. ALGORITHM for STRAIGHT SKELETON

Fig.3 shows the overall outline pseudo-code for the straight skeleton computation by split & edge event and collapse of a triangle to a node. At first, one simple polygon (**P**) is given such as shown in Fig.2. If there is any reflex vertex in the **P**, then it can be divided into two or more polygons.

At four lines from the top of the code, the system calculates e_d_{shri} for all edges and finds the shortest of them. Then, the system checks if split event occurs by increasing d_{shri} by (e_d_{shri} / n_step) . In this way, the shrinking process may proceed until d_{shri} reaches the shortest e_d_{shri} found. In the process, a split event may happen and the polygon will be divided into some polygons: **Ps**. In the upper half of the code (split event process), all divided polygons are checked if they can be divided more. As long as there is some **Ps** that can be divided, split event will continue. After that, the system concentrates on the edge event procedure.

In the split event process, during shrinking to the shortest e_d_{shri} , the system checks if a line segment of an angular bisector from a reflex vertex intersects another edge of the polygon or not. If an edge is found intersected, the system calculates the node position by the split event. However, one edge will be intersected by several angular bisectors from several reflex vertices. Among the several reflex

vertices, the reflex vertex that gives the shortest d_{shri} will be selected for calculating the position.

After any type of event happens and the polygon changes topologically, there remains one or more new split polygons which are shrunk recursively if they have non-zero area. At that moment, the system recalculates the length of each edge and internal angles of each vertex in order to find the shortest d_{shri} for next events.

In the code, the **P** has members: 'split event finish flag' ($sp_ev_fin_fl$) and 'edge event finish flag' ($ed_ev_fin_fl$) which indicate whether or not the **P** can be processed by 'split event' or 'edge event' respectively, during the shrinking process. If ' $sp_ev_fin_fl$ ' is set for the **P**, then the **P** is finished with split event checking. If ' $sp_ev_fin_fl$ ' is reset, then the **P** will be checked whether split event is happened or not.

In the upper half of the algorithm, if at least one possibly divided **P** remains unchecked for 'split event', then 'SplitEventLoopFinish_flag' will be reset and the system cannot get out of the 'while loop'. After all **Ps** have been checked for 'split event', then all **Ps** are checked only for 'edge event' and then 'triangle' procedure for nodes generation as shown in the lower half of the algorithm.

```

While (Event procedure is not finished for all split P) {
  While ('SplitEventLoopFinish_flag' == reset) {
    For all one or more split P: { If (P.  $sp\_ev\_fin\_fl$  == reset) {
      Find the shortest  $e\_d_{shri}$  of the P.
      Check if Split Event occurs by increasing  $d_{shri}$  by  $(e\_d_{shri} / n\_step)$ .
      If (Angular bisector from a reflex vertex intersects
        another edge) {
        Calculate the node position by Split Event.
      }
    }
  }

  For all one or more split P: { If (P.  $sp\_ev\_fin\_fl$  == reset)
    Reset 'SplitEventLoopFinish_flag'
  }
} /* For " While ('SplitEventLoopFinish_flag' == reset) {" */

For all one or more split P: { If (P.  $ed\_ev\_fin\_fl$  == reset) {
  Find the shortest  $e\_d_{shri}$  of the P; Shrink P by  $e\_d_{shri}$ ;
  Calculate the node position by Edge Event.
}

For all one or more split P: { If (P is a triangle) {
  Calculate the node position of the triangle.
  Associate the node with the original edge.
}
}
} /* For "While (Event procedure is not finished for all split P) {" */

```

Figure 3. Algorithm for forming straight skeleton by Split & Edge event and Collapse of a triangle to a node

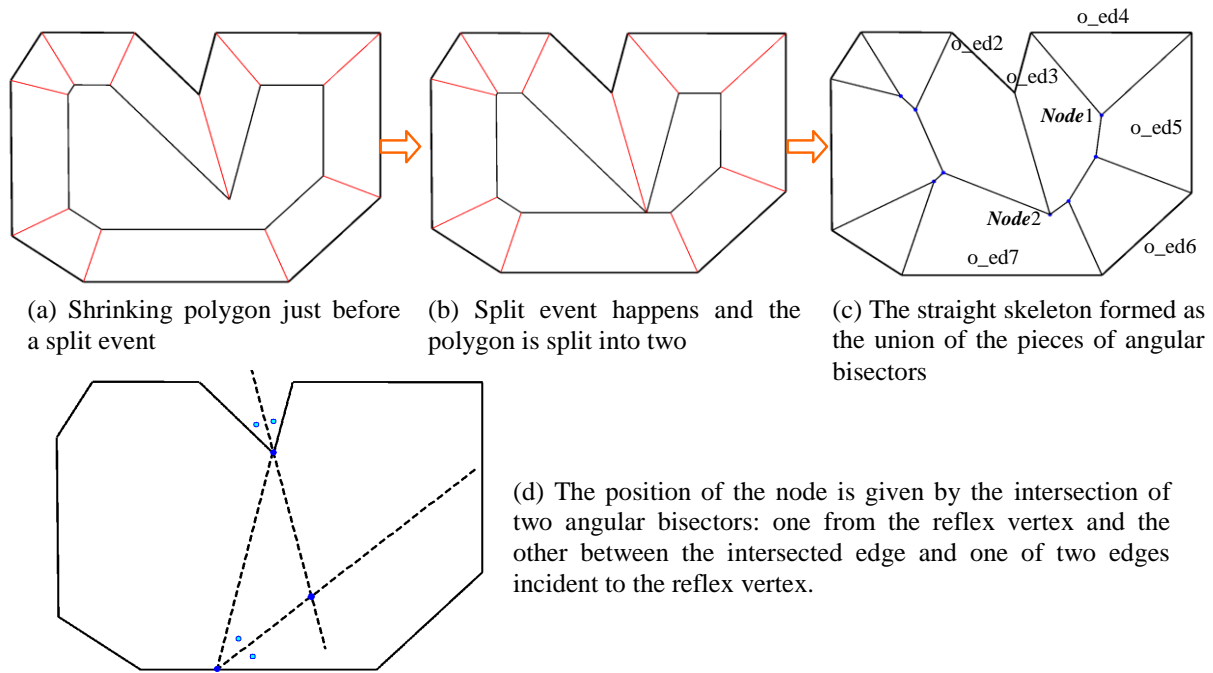


Figure 4. How a split event happens, and how the position of the node is calculated

The generated nodes will be associated with the edges of original P (original edge: o-edge), since at least three original edges sweep to form the node. Therefore, at each event when the node is generated, at least three o-edges will be linked to the node. When a square or a regular hexagon collapses to a node, four or six o-edges will sweep into a node. This is the case of degeneration.

The third event as ‘the simultaneous one’ is processed at ‘edge event’, since the other split polygon disappears into a node in this event. After detecting the split event and edge event have occurred simultaneously, the system deals with the event and links the generated node to three o-edges.

4.2. HOW MONOTONE POLYGONS are FORMED

Fig.4(c) shows how these interior monotone polygons are formed. When a shrinking process starts, the edges of the polygon sweep inwards from their original edge (o-ed_i). Nodes arise from the edge event or the split event. For example, *Node1*, arisen by the edge event, is the convergent point into which consecutive three original edges (from o-ed₃ to o-ed₅) sweep. On the other hand, *Node2*, arisen by the split event, is the intersection point between the angular bisector from the reflex vertex (between o-ed₂ & o-ed₃) and the intersected edge (o-ed₇). Since at least three original edges (o-ed_i) sweep into a node, the node keeps information about which o-ed_i makes up the node itself. In order to form monotone polygons, following the original edge one

by one, the system searches which node has the same original edge number. For example, o-ed₄ has an only one node (*Node1*) that has the same original edge number, whereas o-ed₃ has four nodes that have the same original edge number including such as *Node1*, *Node2*. The nodes belonging to each o-ed_i are sorted according to the coordinate value on the axis parallel to each original edge (o-ed_i) vector. These nodes are coplanar, and will form a roof board for a 3D building model.

Fig.4(d) shows how a split event happens, and how the position of the node arisen by the split event is calculated. The position of the node is given by the intersection of two angular bisectors: one from the reflex vertex and the other bisector between the intersected edge and one of two edges incident to the reflex vertex.

For some polygons in Fig.5 showing a shrinking process of an orthogonal polygon, the event different from the two events mentioned will happen. In this research, it is proposed to add the third event in which a reflex vertex runs into the edge, but the other split polygon is collapsed into a node since an edge event happens in the split polygon at the same time. This event happens at an orthogonal part of the polygon as shown in Fig.5. In the process, as shown in Fig.5(b) & (c), the system detects ‘the third event’ by checking if pt_{ia} (vertex) is on ed_{ib} (edge) or pt_{ib} is on ed_{ia} where pt_{ia} & pt_{ib} are the vertices next to two vertices coherent by the edge event, and ed_{ia} & ed_{ib} are the edges adjacent to these two coherent vertices.

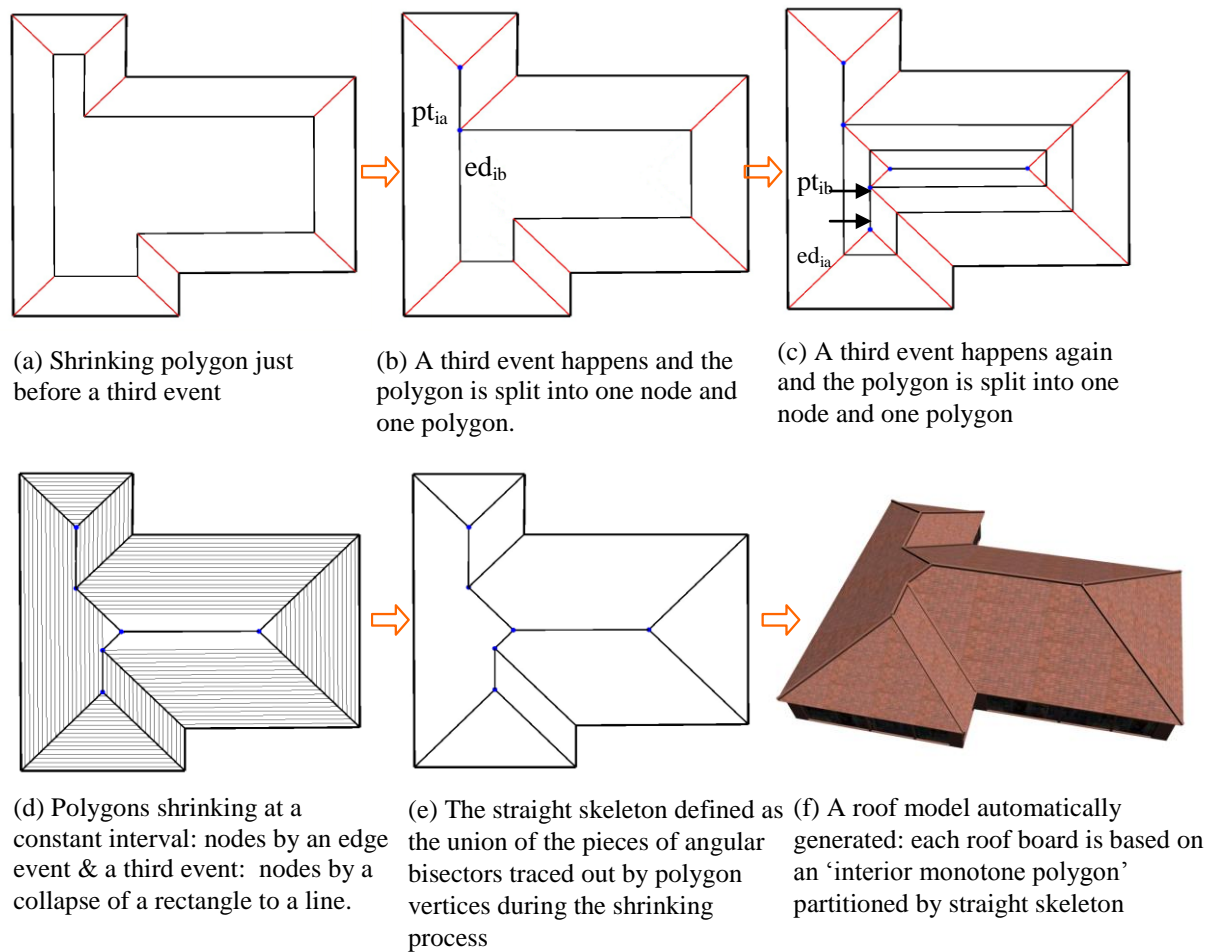


Figure 5. Shrinking process and a straight skeleton for third events

Aichholzer et al. [Aic95a] demonstrated three edge events let a triangle collapse to a point in the last stage of each split polygon as shown in Fig.2(d). In this paper, it is proposed to add the case in which two edge events let a rectangle collapse to a line segment ('a line of convergence') in the last stage, a rectangle whose opposite sides have the same and the shortest e_d_{shri} .

Since a line segment does not have area, it is not shrunk anymore. The central area of an orthogonal polygon in Fig.5(d) shows a line of convergence to which the shrinking polygon (rectangle) is converged.

5. APPLICATION AND CONCLUSION

Here are the examples of 3D building models automatically generated by the integrated system. Fig.6 shows the examples of 3D building models automatically generated by the straight skeleton computation from non-orthogonal building polygons. To ease the discussion, Aichholzer et al. [Aic95a] exclude degeneracies caused by special shapes of polygon, e.g., a regular polygon. In this paper, we deal with the degenerate cases in which more than

three edges are shrunk to a point. Ideally, simultaneous n edge events cause a regular n -gon to collapse to a point but it is difficult to draw such a perfect regular n -gon. Accordingly, the system rectifies the shape of the regular n -gon so as to let n edge events at the same time. Fig.6 center shows the 3D dodecagon building model automatically generated based on the degeneracy of 12 edges being shrunk to one node.

Fig.7 shows proposed digital map for the town and an automatically generated 3D urban model: town houses with doom roofs created by straight skeleton computation in the middle of the image.

For everyone, a 3D urban model is quite effective in understanding what if this alternative plan is realized, what image of a sustainable city will be. Traditionally, urban planners design the city layout for the future by drawing building polygons on a digital map. Depending on the building polygons, the integrated system automatically generates a 3D urban model so instantly that it meets the urgent demand to realize another alternative urban planning for sustainable development.

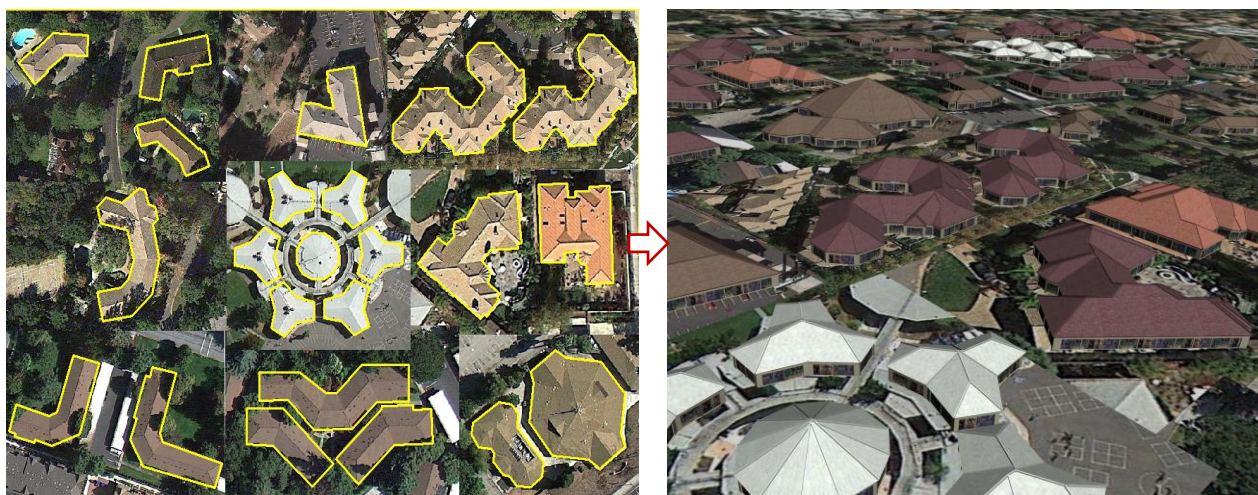


Figure 6. Non-orthogonal building footprints and 3D building models automatically generated by straight skeleton computation

If given digital maps with attributes being inputted, as shown in ‘Application’ section, the system automatically generates two hundreds 3D building models within less than 30 minutes.

In either orthogonal or non-orthogonal building polygons, the new system is proposed for automatically generating general shaped roof models by the straight skeleton computation. In this paper, the algorithm for ‘forming straight skeleton by split & edge event’ is clarified and the new methodology is proposed for constructing roof models by assuming the third event in addition to two events and, at the end of the shrinking process, some rectangles are converged to a line of convergence. Thus, the proposed integrated system succeeds in automatically generating alternative city plans.

The limitation of the system is that automatic generation is executed based only on ground plans or top views. There are some complicated shapes of buildings whose outlines are curved or even crooked. To create these curved buildings, the system needs side views and front views for curved outlines information.

Future work will be directed towards the development of methods for the automatic generation algorithm to model curved buildings by using side views and front views.

6. REFERENCES

- [Aic95a] Aichholzer, O., Aurenhammer, F., Alberts, D., and Gärtner, B.: ‘A novel type of skeleton for polygons’, *Journal of Universal Computer Science*, 1 (12): 752–761 (1995).
- [Car10a] Carlos, V. A., Daniel, A. G., and Bedřich, B.: ‘Building reconstruction using Manhattan-world grammars’, *Computer Vision and Pattern Recognition (CVPR)*, 2010 IEEE Conference on: 358 – 365 (2010)
- [Dan05a] Daniel, B. R., and Daniel, A. G.: ‘Build-by-number: rearranging the real world to visualize novel architectural spaces’, *Visualization*, 2005. VIS 05. IEEE, 143 – 150 (2005)
- [Dan07a] Daniel, A. G., Paul, R. A., and Daniel, B. R.: ‘Style Grammars for interactive Visualization of Architecture’, *Visualization and Computer Graphics*, IEEE Transactions on Volume:13, 786 – 797 (2007)
- [Gru98a] Gruen, A., Wang, X.: ‘CC-Modeler: A topology generator for 3-D city models’, *ISPRS Journal of Photogrammetry & Remote Sensing*, Vol.53, No.5, pp.286-295 (1998).
- [Gru02a] Gruen, A., and et al.: ‘Generation and visualization of 3D-city and facility models using CyberCity Modeler’, *MapAsia*, 8, CD-ROM (2002)
- [Mül06a] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L.: ‘Procedural modeling of buildings’, *ACM Transactions on Graphics*, 25, 3, 614–623 (2006)
- [Nia09a] Nianjuan, J. P. T., and Loong-Fah, C.: ‘Symmetric architecture modeling with a single image’, *ACM Transactions on Graphics - TOG*, vol. 28, no. 5 (2009)
- [Par01a] Parish, I. H. Y., and Müller, P.: ‘Procedural modeling of cities’, *Proceedings of ACM SIGGRAPH 2001*, ACM Press, E. Fiume, Ed., New York, 301–308 (2001)

[Sug09a] Kenichi SUGIHARA: “Automatic Generation of 3D Building Models with Various Shapes of Roofs”, ACM SIGGRAPH ASIA 2009, Sketches DOI: 10.1145/1667146.1667181 (2009)

[Sug12a] Sugihara, K. and Kikata, J.: “Automatic Generation of 3D Building Models from Complicated Building Polygons”, Journal of Computing in Civil Engineering, ASCE (American Society of Civil Engineers), DOI: 10.1061/(ASCE)CP.1943-5487.0000192 (2012)

[Suv02a] Suveg, I., and Vosselman, G.: ‘Automatic 3D Building Reconstruction’, Proceedings of SPIE, 4661, 59-69 (2002)

[Zla02a] Zlatanova, S., and Heuvel Van Den, F.A.: ‘Knowledge-based automatic 3D line extraction from close range images’, International Archives of Photogrammetry and Remote Sensing, 34, 233 – 238 (2002)

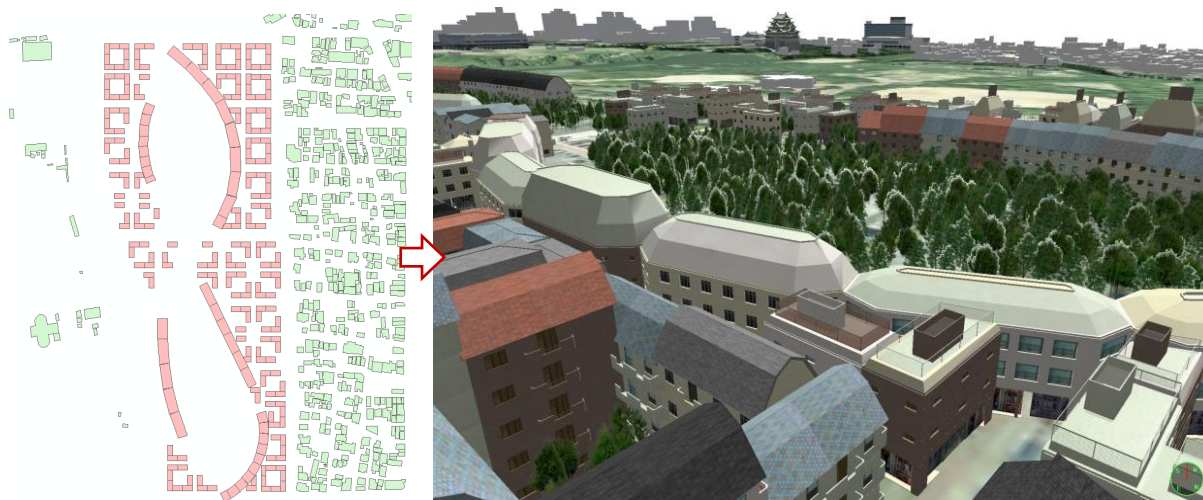


Figure 7. Proposed digital map for the town and an automatically generated 3D urban model: town houses with doom roofs created by straight skeleton computation in the middle of the image

PoolLiveAid: Augmented reality pool table to assist inexperienced players

Ricardo Alves

Institute of Engineering
University of the Algarve,
Campus da Penha
8005-139 Faro, Portugal

ricardo_alves_r16@hotmail.com

Luís Sousa

Institute of Engineering
University of the Algarve,
Campus da Penha
8005-139 Faro, Portugal

luiscarlosrsousa@outlook.com

J.M.F. Rodrigues

Institute of Engineering and
Vision Laboratory, LARSyS,
University of the Algarve,
8005-139 Faro, Portugal

jrodrig@ualg.pt

ABSTRACT

PoolLiveAid is an augmented reality tool designed to assist unskilled or amateur pool, or snooker or billiards players in predicting trajectories. A camera placed above the table acquires and processes the game on-the-fly. The system detects the table border, the ball's position and the pool cue direction in order to compute the predictable trajectory of the white ball, and the ball directly in its trajectory. The output result is then forwarded to a projector, placed above the table, which then projects onto the snooker playable field. A skilled player can also save a specific layout of a move and load it later in order to achieve the best shot and practising.

Keywords

Augmented reality, computer vision, pool game.

1. INTRODUCTION

A pool game can be very challenging and tricky. The first contact with a game of pool can be very frustrating for an unskilled player, requiring many hours of practise to understand even the more basic and classical mechanics that exists in this game.

In this paper we introduce a tool to assist mainly amateur pool players to train themselves by showing them on-the-fly in the pool table what will happen when the white ball is hit, helping the player to make the best decision, thus preventing him from playing countless times before getting it right. On the other hand, a skilled player can also save a specific layout of a move (or a group of moves) and load it later, project it directly onto the pool table in order to achieve the best shot and for practise.

This tool was developed to use the pool table as the interface. The system works with several varieties of tables, regardless of the cloth and colour of the ball (be it pool, snooker or billiard), with any camera that has HD feature, and a projector placed above the table. A camera is placed above the table for capturing and processing the game. The system detects the table border, the ball's position and the

pool cue direction in order to compute the predictable trajectory of the white ball, and the ball directly in its trajectory. The output result is then forwarded to a projector, which then projects onto the snooker playable field.

There are several examples of tools connected, to some extent, to the game of pool, snooker or billiard. Denman et al. [DRK*03] presented three tools applied to footage from snooker broadcasts. The tools allow parsing a sequence based on geometry, without the need for deriving 3D information. They also allow events to be detected where an event is characterised by an object leaving the scene at a particular location. The last feature is a mechanism for summarising motion in a shot for use in a content based summary. Shen and Wu [SW10] also analyse videos. They did an automatic segmentation method of local peak edges to extract the table, and by using several pre-processing, morphological processing, clustering and HSV colour space they detect the ball to produce a 3D reconstruction of the game. Also related to video analysis for a 3D representation with different goals we have [HM07, HGB*10, PLC*11, LPC*11, LLX*12].

On a different level, Dussault et al. [DGM*09], Archibald et al. [AAG*10] and Landry et al. [LDM11] presented a computational system to create a robot capable of selecting and executing shots on a real table. Some of these authors, Leckie and Greenspan [LG06] presented a paper on the physics of the game of pool. One of these authors also has a web page with a tool somewhat similar to ours: ARPool is a projector-camera system that provides

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

real-time feedback to a pool player directly on the surface of the table. However, to our knowledge, there are no publications on this tool (only the web-page). Also related to “robotic pool,” [NKH*11] presented a robot capable of playing on a normal-sized pool table using two arms. The robot can accurately locate the pool table, the balls on the table and the cue, and subsequently plans the next shot. In this case they use a green pool cue an almost white cloth (they also project trajectories on the table). Both robotic systems were tested under laboratory conditions (specific light conditions, etc.).

The main contributions of this paper are a system that: (a) works in real club/pub environment and can be mounted without the need for any changes in terms of table position, lights, etc. Only two supports are needed: one for the camera and one for the projector. (b) Uses the table as the surface for projection and interface with the user and (c) focuses mainly on amateur real players, who are learning to play, or players who want to see on-the-fly a mistake made in a previous play.

In section 2 we present table, ball and cue detection. In section 3 we compute the ball trajectories and explain how to project them onto the table. In section 4 tests and results from the two previous sections are presented. In section 5 we briefly show the main menus of the tool and finally in section 6 we present the conclusions and future work.

2. Table, ball and cue detection

As mentioned in the Introduction, the system was developed based on real pool tables, balls and pool cues. An HD camera was placed over the table, so as to capture the whole table (preferentially in the centre of the pool table). For the images and tests shown in this paper we used a simple HD webcam (around 25 fps), attached to the lamp that was over the table.

The second component of the system is a quality projector (the lighter the surroundings, the better the projector has to be). This can be placed above the table, on the ceiling, projecting over the table, or in a hall near the table (near the ceiling), so the projector can project onto the entire table. In most of our tests, the last situation was the one used, due to the ceiling being too low.

Figure 1, in the top, illustrates the system layout, with the position of the camera and projector. The bottom picture is an example of an image (frame) acquired by the camera. As can be seen, we do not need a perfect image of the table, only an image that catch the entire table.

In the rest of this section we will explain in detail the table boundaries, ball and cue detection.



Figure 1. In the top the system layout, camera and projector position in relation to the pool table. In the bottom, the one acquired image (frame).

2.1.1 Pre-processing: Noise-reduction

A pool player needs to think of his/her next move, so he/she needs to have information on the next possible shot as early as possible, in order to give him/her a perception of what he/she is actually doing or aim to do. Nevertheless, information on the trajectory of a future shot is only needed when and every time movement stops. Based on that, a pool game has two distinct phases of information extraction: (a) detection of any motion, including the pool cue and (b) ball information when the game stops.

One of the main challenges while working with a real pool/snooker room is the noise in the captured frames due to different factors, e.g., the type of lightning. Upon this, we used two different noise-reduction algorithms, depending on which information was to be extracted: (a) motion or (b) balls.

Let be $P_t(x, y)$ the RGB frame acquired in instance t and (x, y) the pixel coordinates within the frame. We used a (a) Gaussian Filter (G) [Rus11], with $\sigma = 2$, when needing to analyse images in real time $G_t(x, y) = G(P_t(x, y))$ and we did a (b) frame time average [Rus11] when ball motion on the table was stopped (for pool table detection and balls detection), i.e.,

$$A_t(x, y) = \frac{1}{N} \sum_{k=t-N}^t P_k(x, y),$$

with N the number of frames to average (in present results $N = 5$) and A_t the average result for instance t .

Figure 2 shows from top to bottom a section of a pool table image. The resulting image after applying the Gaussian filter (G_t), as expected, a blur appeared, and after applying average filter (A_t), we can see the improvement in table cloth.

All parameters used in the pre-processing stage (σ and N) are calibrated only once in the setup stage, for each environment, and can be slightly different for each environment.

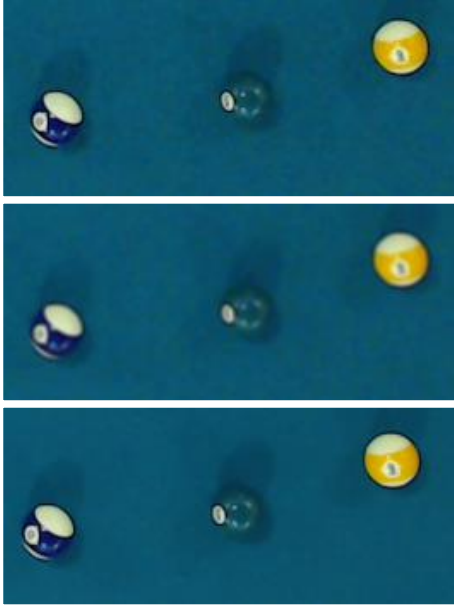


Figure 2. Top to bottom, a section of P_t , the same section after G_t and A_t .

2.1.2 Extracting Tables Boundaries

Extracting table boundaries with precision is extremely important. Trajectories of a ball are directly connected to these boundaries, causing a reflection on the trajectory which will be more perfect the more accurate a boundary is detected. If a boundary is just a few pixels wrong, the resultant trajectory will propagate the error, making it worse as distance to this boundary increases.

As long as a camera is fixed, boundaries never change (the table position never changes during a game). Based on this, they are only calculated once in the initial setup of the system.

Starting with an empty table, and from the middle of the image $A_t(x, y)$, which will or should be the centre of the table, going through top, bottom, left and right, these lines are extracted through edge detection using the Canny edge detector [Can86] (with $\sigma = 3$, $T_l = 0.1$ and $T_h = 0.5$) in $E(x, y)$ being $E = \frac{1}{5} \sum_{t=5}^{10} A_t$ (the average of the initial 10 images

acquired in the setup procedure), followed by the Hough Transform [DH72].

The results of a Hough transform are a set of candidate lines for the table's boundaries shown in Fig. 3 top. Every line detected this way is tested, checking if its angle is near $0^\circ (\pm T_{th})$, in case it is a top or bottom table boundary, or if its angle is near $90^\circ (\pm T_{th})$, in case it is a left or right table boundary, with $T_{th} = 1^\circ$. If a line succeeds this test, it is checked if there are other possible table boundaries near it. Inclination test is applied to every line near the first one detected, and if it succeeds, the average line of all lines detected will be considered to be a table boundary. After all this, with linear equations, all 4 corners of the table are found, see Fig. 3 bottom.

If necessary, only if the automatic boundary detection doesn't work perfectly, we allow for the possibility in the setup procedure for the boundaries to be manually adjusted (slightly), using the setup menu interface and computers mouse.

If the camera is not in the centre of the table, or if T_{th} necessary to detect the lines, or the final computed line is bigger than 1° , then a perspective transform (e.g. [Rus11]) from the original image (video frame) to a "model pool table" is required. This will decrease the performance of the system very slightly, once every operation (ball detection, trajectories, etc.) has to be affected by the same transformation. To simplify the explanation in the following sections and for the rest of the paper we ignore this transformation (i.e., we consider $T_{th} \leq 1^\circ$).



Figure 3. Top, detected lines and in the bottom final automatic table boundaries extraction.

2.1.3 Movement Detection

After the detection of the table boundaries, movement detection is one of the more important functions to be computed. As previous explained, balls are only detected when movement on the table stops, avoiding the program to make unnecessary computations for the detection of the balls (see Section 2.1.5).

Movement detection also has a second purpose: pool cue detection (see Section 2.1.4). Obviously, the cue detection is more reliable if there were no balls on the table. As this is not possible, every time movement stops, a frame $G_{rfc}(x,y)$ is taken by the camera, allowing it to be a reference for what it is in the table at that moment, we remember that in a pool/snooker game the player has a penalty if it touches or moves any ball. The G_{rfc} frame is then going to be used as reference (“ground-truth”) for the cue detection.

Movement detection is based on the subtracting the actual frame with the previous frame,

$$D_t(x,y) = G_t(x,y) - G_{t-1}(x,y).$$

The output (D_t), is converted to grayscale (Dg_t), and a threshold with the goal of creating a binary image is applied. All pixels with the level of gray above T_g are assigned to 0 otherwise to 255, returning Dgb_t . The value of T_g is automatically computed by calculating the maximum value that the histogram of $E(x,y)$ during the setup stage changes more than 0.003% of the total of element in the playable field (we use 25 in the examples presented).

After this all pixel inside the table playable field (mark by the 4 red lines in Fig. 3 bottom) are counted, C_t , if there are less then 0.003% of pixels with 255, it is considered that there is no more movement on the table, and the counted pixels are due to noise,

$$C_t = \sum_{x=0}^W \sum_{y=0}^H c_t(x,y),$$

$$\text{with } c_t(x,y) = \begin{cases} 1, & \text{if } Dgb_t(x,y) = 255 \\ 0, & \text{otherwise,} \end{cases}$$

and W e H the weight and height of the playable field.

In summary, if $C_t < 0.00003 \times W \times H$, there is no movement in the table, turning a movement flag OFF, otherwise turning it ON. Every time there is movement on the table, like e.g., cue striking a ball this flag is put to ON, but if set to OFF (by the above process) then we can conclude that movement on the table has stopped, a reference frame G_{rfc} is taken and triggering ball detection algorithm.

2.1.4 Cue Detection

The cue information is only needed when the cue is relatively close to the white ball, as the cue is always

placed close to it when a player is preparing to strike the white ball. As result, information extracted is only considered if close, in a circular area, to the white ball.

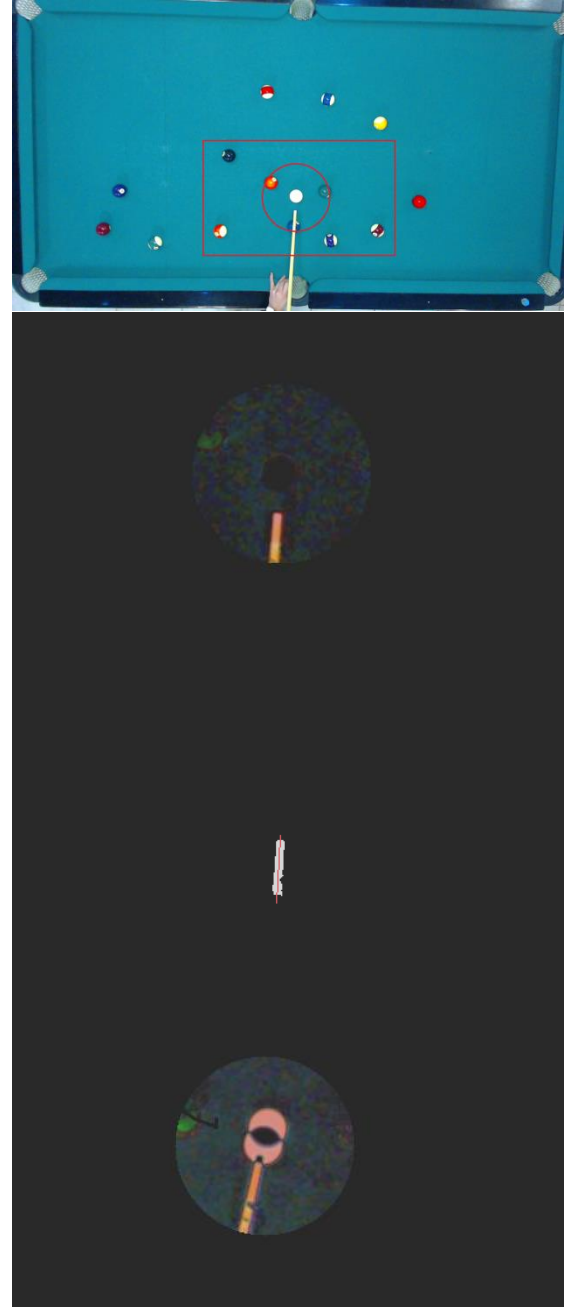


Figure 4. Top to bottom, frame with the RoIs represented, image after subtraction with G_{rfc} , the cue line detection and shot detection.

For the cue detection, the (a) current frame G_t , Fig. 4-1st row, is subtracted to the reference frame G_{rfc} , Fig. 1-2nd row, enabling the balls not to interfere with the cue detection. This difference is only done in a (b) small square region of interest (RoI) (to save CPU time) with dimension $M \times N$ pixels, the

differences of the images are shown in Fig. 4-2nd row. For the final cue validation, a player hand must not be close to the white ball and for this reason we consider (c) a 2nd circular RoI, radius of $R_c = 4 \times r$ pixels, being r the radius of the white ball (see section 2.1.5 for r calculation). Following the same principle $M = 12 \times r$ and $N = 6 \times r$. In this circular RoI usually no player put he's hand, and if he put his hand in this area it is considered not the correct way to hold the cue (the system detect if there is a hand in this area and alert the user).

Being $DG_t = |G_t - G_{rfc}|$, and from the converted grayscale of $\sum_r DG_t$, it is applied a (d) threshold T_r (we used $T_r = 50$, this value was computed empirically) to binarise the result, obtaining a white shape Fig. 4-3rd row.

From this shape (e) we compute the middle line (the line that splits this shape into two), red line in Fig. 4-3rd row. This line is considered to be the line of the cue, where the tip of the cue is considered to be the point of the line closer to the white ball.

When a player shoots the white ball, the program needs to stop detecting the cue, in order to stop to show trajectory lines (see section 3). This is achieved, once again, by comparing the actual frame $G_t(x, y)$ with the reference frame G_{rfc} .

Every time a new frame is acquired, an absolute subtraction is made (DG_t), in order to test what happens in the circular ball area. If the white ball is not stroke then the pixels value in the white ball area are (near) 0, since those areas of the images are equal, but if the white ball starts moving, then this area starts getting values different from 0, see Fig. 4 bottom.

Since the cue, can also be placed in contact and above the white ball, it's important to choose number of pixels that defines the white ball to be in movement (B_t),

$$B_t = \frac{1}{\pi r^2} \sum_{x=x_0-r}^{x_0+r} \sum_{y=y_0-r}^{y_0+r} |G_t(x, y) - G_{rfc}(x, y)|,$$

within $(x - x_0)^2 + (y - y_0)^2 \leq r^2$, being (x_0, y_0) the centre of the circular RoI. We use $B_t = 15$, to consider the ball in motion, see Fig. 4 bottom, once again this can be configured in the setup procedure, plus, depending of the camera used it is possible to compute de velocity of the strike (not implemented yet).

2.1.5 Detecting and Identifying the Ball

Every time movement stops, the ball's detection starts, consisting in comparing the actual average frame $A_t(x, y)$ with the initial average frame that contains nothing but the empty table $E(x, y)$ (Fig. 5, 1st and 2nd row respectively),

$$F_t(x, y) = |A_t(x, y) - E(x, y)|,$$

after which a binarisation is applied. Again, as in section 2.1.3, $F_t(x, y)$ is converted to grayscale $Fg_t(x, y)$ and every pixel with a value above 15 is put to white, obtaining $Fgb_t(x, y)$, see Fig. 5-3rd row.

Using a contours finder [Rus11], we can know find various blobs which may, or not, be balls.



Figure 5. Top to bottom, A_t , the reference frame E , the subtracted binary image from the above images Fgb_t and the images with the detected balls in red, plus the white.

Every blob detected is then considered to be a ball if it meets the three following parameters: (a) Ratio

(R_1) between blob's height (Hb) and width (Wb) is approximately 1,

$$R_1 = Wb/Hb \approx 1.$$

(b) Relation between blob's area and circle area, both approximately equal, $\pi \times r^2 \approx A_b$, with A_b the area of a circle, with (x_b, y_b) the circle centre and r ($\approx Wb$) the radius of the circle,

$$A_b = \sum_{x=x_b-r}^{x_b+r} \sum_{y=y_b-r}^{y_b+r} a_b(x, y)$$

$$\text{with } a_b(x, y) = \begin{cases} 1, & \text{if } Fgb_t(x, y) = 255 \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{within } (x - x_b)^2 + (y - y_b)^2 \leq r^2.$$

(c) Ratio (R_2) between a circle and square area,

$$R_2 = \frac{\pi r^2}{(2r)^2} \approx \frac{\pi}{4},$$

the last condition compares if the blob area verifies the circle and square areas relation. If these three relations are true, then the blob detected is considered to be a ball.

After detecting all the blobs, we must distinguish which of the blobs is the white (BW). This is achieved if both extracting contrast information (BC) and a ratio (BR) occur:

(a) For the contrast we check all the blobs (i) which is the one most differed from the background

$$BC = \max_i \left[\frac{1}{A_{b,i}} \sum_{x=x_{b,i}-r}^{x_{b,i}+r} \sum_{y=y_{b,i}-r}^{y_{b,i}+r} F_{t,i}(x, y) \right],$$

$$\text{with } (x - x_b)^2 + (y - y_b)^2 \leq r^2.$$

(b) The ratio (BR) between bright area and blob's area detected,

$$BR = \max_i \left[\frac{1}{A_{b,i}} \sum_{x=x_{b,i}-r}^{x_{b,i}+r} \sum_{y=y_{b,i}-r}^{y_{b,i}+r} br(x, y) \right],$$

$$\text{with } br(x, y) = \begin{cases} 1, & \text{if } Ag_t(x, y) \geq T_{BR} \\ 0, & \text{otherwise,} \end{cases}$$

within $(x - x_{b,i})^2 + (y - y_{b,i})^2 \leq r^2$. The Ag_t is the A_t converted to grayscale, and the T_{BR} was computed empirically. Once again this can be changed in the setup procedure of the system, but for all the tests done we always used $T_{BR} = 200$.

The blob with the brightest area is the white ball, $BW = BC \cap BR$. Blobs that are not white will have few pixels in the bright area, while a white ball will have more pixels in the bright area.

As result of this operation, we obtain all balls detected with the white ball being distinguished from all the other as shown on Fig. 5 bottom. If we want to know which ball we are playing against, or make an automatic table of scoring (in the case of snooker), we can apply the above two principles (without applying the max) creating two tables were the colour are ordinates, this can also be complemented using thresholds in HSV colour space.

3. Ball Trajectories

Ball trajectories shown in real time (see Fig. 7) can be computed after the cue and the white ball centre was detected, using simple and well known math formulas. The cue stick is represented by two points, and we can compute the correspondent line equation ($m \times x + b$) and the white ball by its radius r and centre point (x_o, y_o) .

The white ball is only going to be shot if the cue line intercepts any point of the white ball contour, i.e., if points $P_{int} = (P_{x,\pm}, P_{y,\pm})$, with

$$P_{x,\pm} = \frac{\pm(\sqrt{-b^2 - 2b(mx_o - y_o) + m^2(r^2 - x_o^2) + 2mx_o y_o - y_o^2 + r^2}) - bm^2 y_o + x_o}{m^2 + 1},$$

and

$$P_{y,\pm} = \frac{\pm(\sqrt{-b^2 - 2b(mx_o - y_o) + m^2(r^2 - x_o^2) + 2mx_o y_o - y_o^2 + r^2})m + b + m(my_o + x_o)}{m^2 + 1},$$

the points that contacts the white ball circular surface with the line of representing the cue. Thus, if any of those points (P_{int}) are true, we can start calculating what would be the predictable trajectory, assuming the player will always try to hit its centre. It can be easily calculated applying the cue vector to its centre.

All the different effects that a (semi-)professional player can do it is not considered for the module of the ball trajectories of the tool. This is a tool designer for beginners and they "just want" to hit the white ball to go in a specific direction.

3.1.1 Reflection of the Ball-table

Having detected contact between the cue and ball, the table's reflection can be determined by simple vector maths. As shown in Fig. 6 top, \vec{I} being incident vector, we can obtain \vec{N} , which is vector normal to incident plane,

$$\vec{N}(x, y) = \vec{v}(-y, x),$$

to achieve desired vector reflection it is used

$$\vec{R} = 2 * \vec{N}(\vec{I} \cdot \vec{N}) - \vec{I}.$$

As the white ball centre never intercepts the table's boundaries, every reflection needs to be calculated using an auxiliary boundary moved the ball radius to the centre of the table, giving the result shown on Fig. 6 top.

3.1.2 Balls Interactions

Balls collisions are calculated using vectors. A vector containing the trajectory of a ball is applied to the tangent points of that ball, giving line $L3$ and $L4$ of Fig. 6 bottom. There are two interface points that we need to calculate before we can know what will happen to the intercepted ball.

First we need to know which of these lines, $L3$ and $L4$, intercepts P_{int} of other ball and its interception

point, given by point a of Fig. 6-bottom. The second point is easily given by the trajectory's normal $\vec{N}(x,y)$ which is applied to the length of the intercepted ball radius to its centre, given by point b of Fig. 6-bottom.

The final trajectory, $L2$, is the normal of the vector given by point a and point b applied to the centre $C2$. The point d will make the contact with the point c .

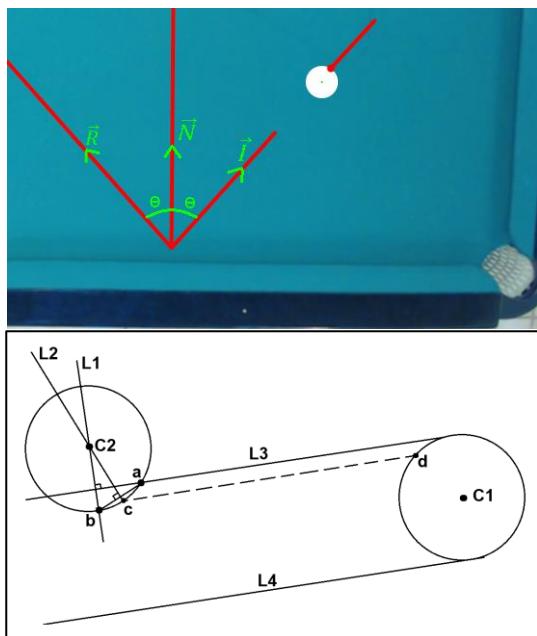


Figure 6. Table's reflection with vectors in the top and collision between two balls in the bottom.

3.1.3 Projecting Ball and Trajectories to the Table

Having all the balls detected and trajectories computed, the next step is to project everything onto the pool table. As mentioned before, we used a projector (see Fig. 1 top), basically matching the table dimensions in pixels (W, H) to the maximum resolution of the projector used. In other words, it is necessary to convert the computed trajectory and the ball's points to the table coordinates, applying a transformation matrix.

After this we create a back image where we render the balls positions; the white ball marked by a circle and the different trajectories. This can be seen in the computer in the background of Fig. 9 top.

Different options can be used, as for example using different colour for different trajectories, a "red colour" for the easiest ball to put in the hole, or compute automatically here to put the cue if we want to put ball x in the side pocket y , save a game, or project onto the table the whole game (or part of the game), project a single continuous image with the ball position stored, to play again (and again) the

same move (this mainly for (semi-)professionals), etc.

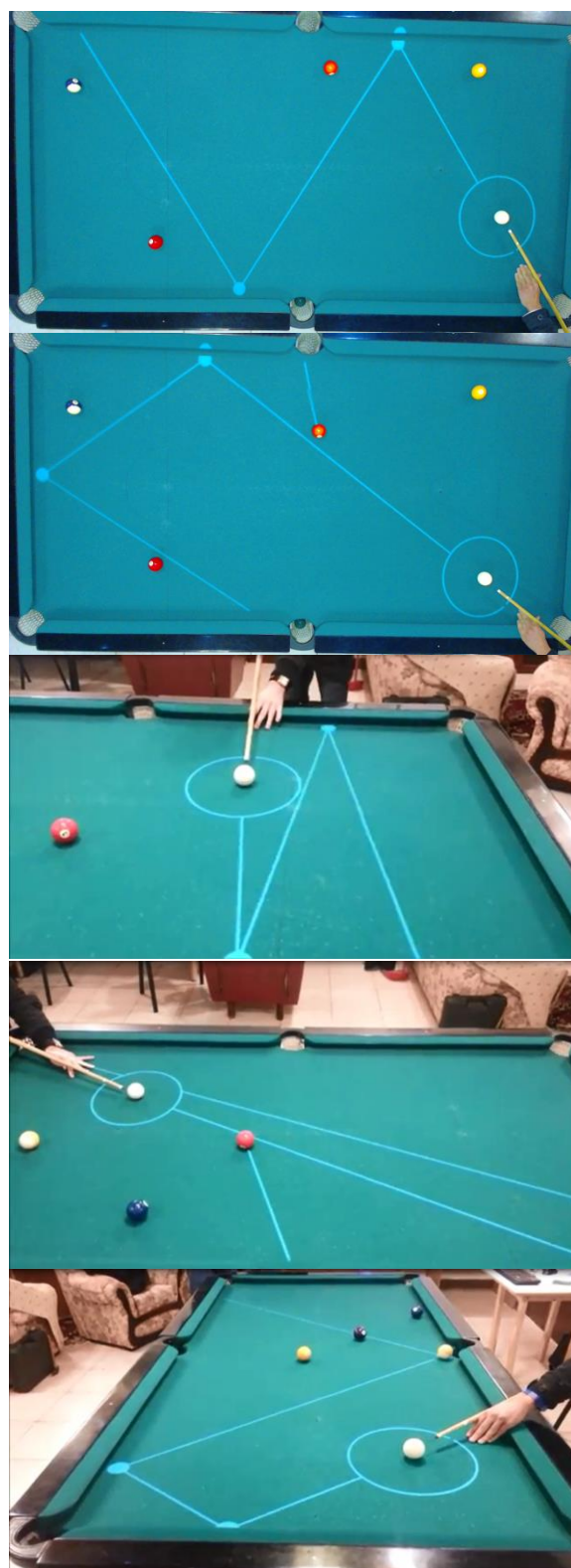


Figure 7. Some results of detected balls and proposed ball trajectories.

Some of options available are shown in the

Augmented Reality Menus section (Section 5).

4. Tests and Results

Figures 7 and 9 show some examples of proposed trajectories. To test the system we invited two very inexperienced players (one male and one female) and we computed around 10 continue minutes of playing, at the same time doing the ground-truth of each play, and we analysed 3 main topics: (a) table boundaries detection, (b) detected balls and (c) expected trajectories.

4.1.1 Table Boundaries Detection

In this first test, we used two different tables in two different rooms, and we changed the lightning, turning on and off all different lights existing in the surroundings. As boundaries are usually the same colour as the table, there is almost no contrast between them, making it difficult to detect if the light in the surroundings is poor.

In well-lit surroundings, by “well-lit” we mean all the usual lights turned on, the table boundaries were always automatically detected (100% of the times) with less than 3 pixel error that can be corrected with the computer mouse.

4.1.2 Ball Detection

Ball detection works very well when balls are not close to each other, with tiny errors on some balls that could not be measured. In a total of 194 balls detection test we obtained 0 false positives and 186 balls successfully detected (96%), where the 8 balls that were not detected were due to being too close to each other (in contact). Also, in this test, all the white balls were successfully detected with zero false positive white balls.

4.1.3 Trajectories

Trajectories predicted are related to the distance a ball can travel and how many bounces they have on a table's boundary. Direct balls were successfully tested in 97% of the cases, where the 3% were due to some imperfections on the cue detection and due to the distance that the white ball would travel being high.

Balls that would bounce on one table were successfully tested in 77%, where the unsuccessful trajectories were due to the distance travelled, cue detection imperfections and spin gained or lost (due to speed) in a table boundary.

Similarly, only 54% of the balls travelled the predicted trajectory when bouncing twice on the table boundaries, due to what we previously stated. Interaction and reflection on other balls were 54% successful due to the errors previously stated and, possibly, to minimum errors on the ball positions.

5. Augmented Reality Menu

After setting up the system, done only once after installing the system, the tool is ready to work.

The users can previously upload photos, or the system can download them directly from Facebook (given authorization by the user). The game starts by projecting the photos over the table (Fig. 8-1st row) when the two players put one hand over the photos an augmented reality pop-up menu appears (Fig. 8-2nd row). Balls can be placed on the table at any moment.

Three main features are shown (from left to right): (a) play with help, as shown in Fig. 7, (b) reload a previous play and (c) save a game/play. Several other menus are available or under construction. When we put the hand over the icon more than 3 second the option is activated, Fig. 8-3rd row, notice the arrow over the hand.

A player can use in-game features that enable the player to save a clip of his last move (option b), in order to see what went right or wrong, and show it on the table on-the-fly. A move can also be saved (option c), in XML format for it to be loaded later, allowing the player to practise that move later.

The hand detection is based on the same principle of the ball detection. It is detected by comparing actual frame A_t with the table reference frame E . If the number of white pixels (PP) is higher than 95% of the icon circular area, then it assumes the player has selected the menu (this has to occur during 3 continuous seconds),

$$PP = \sum_{x=W_m/2-R_m}^{W_m/2+R_m} \sum_{y=H_m/2-R_m}^{H_m/2+R_m} pp(x, y),$$

$$\text{with } pp(x, y) = \begin{cases} 1, & \text{if } Fg_t(x, y) \geq T_m \\ 0, & \text{otherwise,} \end{cases}$$

within the circular area $(x_j - \frac{W_m}{2})^2 + (y_j - \frac{H_m}{2})^2 \leq R_m^2$. W_m and H_m are the dimensions of each icon of the menu, $R_m = \max\{\frac{W_m}{2}, \frac{H_m}{2}\}$ and (x_j, y_j) the centre of each icon; $T_m = 25$.

The same process is used to disable the pop-up menu. If all icon areas have a PP value above T_m for more than 10 second the menu is disabled. To pop-up the menus again, we have to put both hands again in the area where the faces are presented at the beginning.

Despite this menu being quite easy to use, especially for people used to handling tablets, for instance, it can, however, be difficult for people not used to ICT. Taking this into account we are also studying the integration of previous works [SRB09, SFT*13] in hand and head gestures as interface to the tool.

6. Conclusion and Future work

In this paper, we presented a system that aids a beginner player to play pool. Using a standard HD

webcam, it allows detecting table boundaries, balls and cue stick. A projector, placed above the table, can show, in real time, the computed trajectory line in order to give a player a perception of what is going to happen in that particular move.

The system has two stages: (a) The setup, this is done only once, the first time that the system is mounted (or if the table changes position). In this stage there is a computer interface menu with all the parameters that can be adjusted. If any parameter adjustment is necessary, this is done only at this stage. After this (b) the running stage, every interaction with the system is done using the table as interface, i.e., using the augmented reality menu. There are no parameters to be adjusted at this stage.

The system works in real time, and all the tests and results showed were very good. In term of comparison with previous systems, it is quite difficult, because as for the best of our knowledge there isn't any database or ranking to test this algorithms, plus this is only system working in real time in real conditions, systems like [DRK*03, HM07, SW10, HGB*10, PLC*11, LPC*11, LLX*12] work on video taken from championship of pool or snooker, and [DGM*09, AAG*10, LDM11, NKH*11] work in a more or less controlled environment because of the robots.

We must also make a small note about the projector calibration. This is easily done using the menu of the projector itself, keeping only attention to the fact that the projected area must cover all playable area of the table. The projection in the table does not affect the balls and cue detection, as there is no projection when a move is made and only are projected again when all balls stopped (or when the player requests the menu, as referred in section 5). All frame acquisitions are done during these intervals.

In the near future we plan to enhance the ball detection, enabling it to detect balls when they are in contact with each other. We hope to enhance the cue detection, to develop a system to minimize the angular error on a table boundary and to reduce the error given by a player not shooting the ball in the centre of it, taking the speed of the strike into account. Also to improve the quality distortions in the camera optics should be handled. A further future goal is to finish (increase the number of options) and improve the augmented reality menu.

7. ACKNOWLEDGMENTS

This work was partly supported by the Portuguese Foundation for Science and Technology (FCT), project PEst-OE/EEI/LA0009/2011. We also thank Conceição Bravo for the English revising of the

paper and the Association Jovem Sambrasense for providing the pool table for the system development.

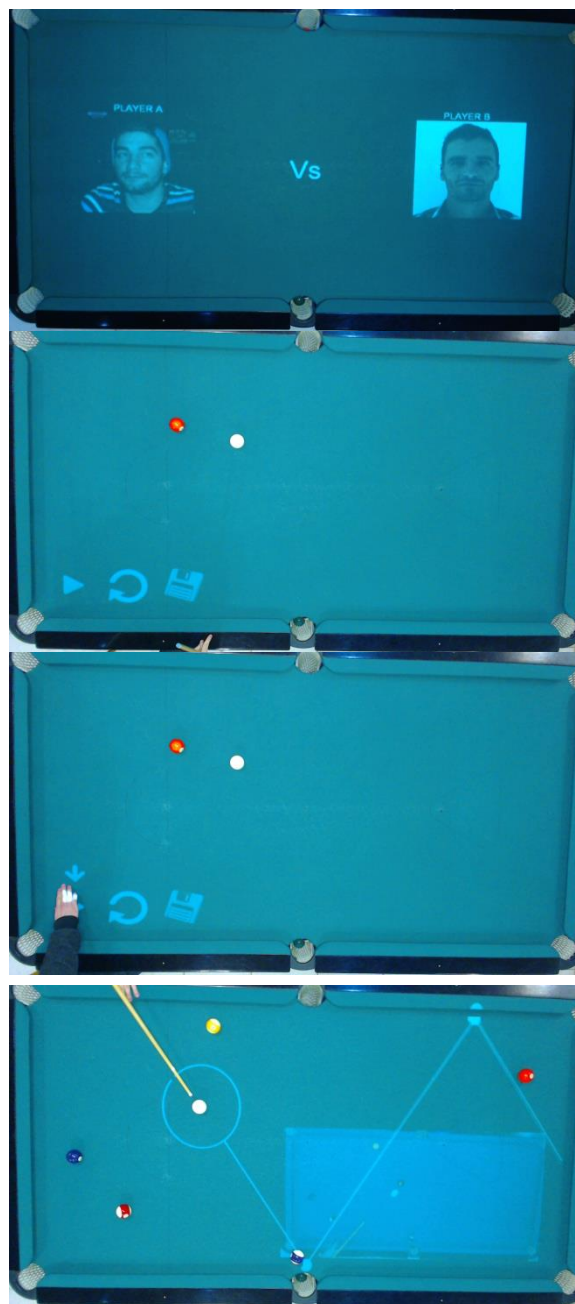


Figure 8. Some examples of the Augmented Reality Menu.

8. REFERENCES

- [AAG*10] Archibald, C., Altman, A. and Greenspan, M. and Shoham, Y. Computational Pool: A New Challenge for Game Theory Pragmatics. Magazine article from AI Magazine, vol. 31, no. 4, pp. 33-41, 2010.
- [Can86] Canny, J. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell., vol. 8, no. 6, pp. 679-698, 1986.

- [DRK*03] Denman, H., Rea, N., Kokaram, A. Content-based analysis for video from snooker broadcasts, *Computer Vision and Image Understanding*, vol. 92, no. 2–3, pp. 176-195, 2003. doi:10.1016/j.cviu.2003.06.005.
- [DH72] Duda, R., Hart, P.: Use of the Hough transform to detect lines and curves in pictures. *Comm. ACM*, vol. 15, pp. 11-15, 1972.
- [DGM*09] Dussault, J., Greenspan, M., Landry, J., Leckie, W., Godard, M., Lam, J. Computational and Robotic Pool. In Chap. XII of *Digital Sport for Performance Enhancement and Competitive Evolution: Intelligent Gaming Technologies*, IGI Global, pp. 194-209, 2009. doi: 10.4018/978-1-60566-406-4.ch012
- [HM07] Hao, G., MacNamee, B. Using Computer Vision to Create a 3D Representation of a Snooker Table for Televised Competition Broadcasting. In *Proc. 18th Irish Conf. on Artificial Intelligence & Cognitive Science*, 2007.
- [HGB*10] Höferlin, M., Grundy, E., Borgo, R., Weiskopf, D., Chen, M., Griffiths, I.W., Griffiths, W. Video Visualization for Snooker Skill Training. *Comput. Graph. Forum*, vol. 29, no. 3, pp. 1053-1062, 2010.
- [LDM11] Landry, J. and Dussault, J. and Mahey, P. Billiards: an optimization challenge, In *Proc. 4th Int. Conf. on Computer Science and Software Engineering*, Montreal, Quebec, Canada, pp. 129-132, 2011. doi: 10.1145/1992896.1992912
- [LG06] Leckie, W. and Greenspan, M. An event-based pool physics simulator. In *Proc. 11th Int. Conf. on Advances in Computer Games*, Taipei, Taiwan, Springer-Verlag LNCS 4250, pp. 247-262, 2006. doi: 10.1007/11922155_19
- [LPC*11] Legg, P.A., Parry, M.L., Chung, D.H.S., Jiang, R., Morris, A., Griffiths, I.W., Marshall, D., Chen, M. Intelligent filtering by semantic importance for single-view 3D reconstruction from Snooker video. In *Proc. 18th IEEE Int. Conf. on Image Processing*, pp. 2385-2388, 2011. doi: 10.1109/ICIP.2011.6116122
- [LLX*12] Ling, Y. Li. S., Xu, P. Zhou, B. The detection of multi-objective billiards in snooker game video. In *Proc. 3rd Int. Conf. on Intelligent Control and Information Processing*, pp. 594-596, 2012.
- [NKH*11] Nierhoff, T., Kourakos, O., Hirche, S. Playing pool with a dual-armed robot," *Robotics and Automation (ICRA)*, *Proc. IEEE Int. Conf. on Robotics and Automation*, pp.3445-3446, 2011. doi: 10.1109/ICRA.2011.5980204
- [PLC*11] Parry, M.L., Legg, P.A., Chung, D.H.S., Griffiths, I.W., Chen, M. Hierarchical Event Selection for Video Storyboards with a Case Study on Snooker Video Visualization, *IEEE Tr. on Visualization and Computer Graphics*, Vol.17, no.12, pp. 1747-1756, 2011. doi: 10.1109/TVCG.2011.208
- [SRB09] Saleiro, M., Rodrigues, J. and du Buf, J.M.H. Automatic hand or head gesture interface for individuals with motor impairments, senior citizens and young children. In *Proc. Int. Conf. on Software Development for Enhancing Accessibility and Fighting Info-exclusion*, pp. 165-171, 2009.
- [SFT*13] Saleiro, S., Farrajota, M., Terzic, K., Rodrigues, J.M.H., du Buf, J.M.H. (2013) A biological and realtime framework for hand gestures and head poses, accepted for 15th Int. Conf. on Human-Computer Interaction - Universal Access in Human-Computer Interaction Conf., 2013.
- [SW10] Shen, W., Wu, L. A method of billiard objects detection based on Snooker game video. In *Proc. 2nd Int. Conf. on Future Computer and Communication*, vol. 2, pp. 251-255, 2010. doi: 10.1109/ICFCC.2010.5497393
- [Rus11] Russ, J.C. *The Image Processing Handbook*, 6th Ed., CRC Press Inc., 2011.



Figure 9. More results with different layouts.

The Retrieval of NURBS-surface by Genetic Algorithm on the Basis of Point Cloud

Eugene Vladimirovich Popov
Professor of NNSACEU
Ilyinskaya Street 65
603950, Nizhny Novgorod,
Russia
popov@sandy.ru

Sergej Igorevich Rotkov
Professor of NNSACEU
Ilyinskaya Street 65
603950, Nizhny Novgorod,
Russia
rotkov@nngasu.r

ABSTRACT

The approach to the geometrical modeling problem solution is described in this report. The approach is dedicated to the approximation of the cloud of points by a NURBS-curve or NURBS-surface and is based on the inheritance mechanism or on the so-called Genetic Algorithm. Genetic Algorithm is the heuristic search and optimization technique that mimics the process of natural evolution. The mechanisms of evolution seem well suited for some of the most pressing scientific problems in many fields. Therefore, the concept of evolution can be applied to solve different computational problems and NURBS-surface retrieval including. The efficiency of the approach is demonstrated by the retrieval of a human face and ship hull surface.

Keywords

Genetic algorithm, NURBS-surface, ship hull design, point cloud retrieval

1. INTRODUCTION

The Genetic Algorithm (GA) is a modern adaptive technique for the solution of functional optimization problems that are frequently used nowadays. It is based on the genetic pattern of biological organisms, namely biological populations that evolve over generations subjecting to the laws of natural selection and the principle "the fittest survive" formulated by Charles Darwin [1]. Similar to these processes the genetic algorithm is able to "solve" real-world problems, if they are properly coded [2]. For example, the GA can be applied to the design of the bridge to find the maximum strength/weight ratio or to determine the most economical pattern when cutting out the fabric cloth. Another example is searching for a set of rules or equations that will predict the ups and downs of a financial market, such as that for foreign currency. Usually such search problems can benefit from an effective use of parallelism, in which many different possibilities are explored simultaneously in an efficient way.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Genetic algorithm (GA) was invented by John Holland [3] and it is still thoroughly investigated in many studies. In contrast to the natural evolution of living organisms the GA only simulates evolutionary processes in populations that are essential for their development. However, there is not any exact answer to the question which biological processes are essential for the development, and which are not [3].

In wilderness individuals are competing with each other within the population for a set of resources such as food or water. Besides, usually members of the population of one species compete for a mate. Those individuals who are best suited for the environment will be relatively more successful to produce offspring [1]. Poorly adapted individuals either will not produce offspring or their offspring will be very few [3]. It means that the genes of highly suited or adapted species will be available in an increasing number of children of each successive generation. Therefore, a new species will become more and more adaptable to the environment [4].

GA uses a direct analogy of such a mechanism. It works with a group of "individuals" (or "genome") i.e. with a population. Each individual represents a possible problem solution and is assessed according to its "fitness" to how "good" the corresponding problem solution is [3]. In nature a "good solution" means that an individual is capable of competing for resources in the most efficient way [1]. The fittest individuals have the opportunity to "reproduce" the breed by "crossover" with other species of the

population. This leads to the appearance of new species that combine some of the characteristics inherited from their parents. The least adapted individuals are less likely to be able to produce offspring, which in its turn leads to the gradual disappearance of their quality from the population in the process of evolution [1], [3].

In this way the whole new population of feasible solutions is reproduced by selecting the best representatives of the previous generation, crossing them and getting a new set of individuals. This new generation has better characteristics than the good members of the previous generation. Therefore, good features are distributed throughout the population from generation to generation. Hybridization of the fittest individuals leads to the fact that the most promising areas of the search are explored. Eventually, the population converges to an optimal solution [1].

2. THE SURFACE RECONSTRUCTION AND GA

Many practical surface reconstruction techniques based on measured data points require the solution of optimization problems in fitting surface data. In general, we need determining the necessary and sufficient conditions for the possibility of GA application to this problem solution.

Let us suppose there is a finite set of geometric parameters $G = \{g_1, g_2, \dots, g_n\}$. Let us also assume that a given finite set of conditions $C = \{c_1, c_2, \dots, c_k\}$ exists. Then we will need to build another finite set of geometric parameters $T = \{t_1, t_2, \dots, t_m\}$ that satisfies each element of the set of conditions C . The algorithm of finding set T is unknown. We should also find such finite set \tilde{T} the elements of which are the set of geometrical parameters \tilde{T}_i that meet *at least one condition* of set C . If they meet *all the conditions* of set C the problem is solved. Therefore, set \tilde{T} should meet the following requirements to solve the problem:

- a certain numerical value q_i must be assigned to each element of set \tilde{T} (\tilde{T}_i) that should indicate the degree of satisfaction of this element to the set of conditions C ,
- the *mutation* operation over any element of set \tilde{T} must be defined (see [1], [4]). This operation is aimed at mapping the required element to another element from the same set \tilde{T} ,

- the *crossover* operation must be defined between any two elements of set \tilde{T} [4] the result of which is also an element of set \tilde{T} .

The finitude of set \tilde{T} is a necessary condition, but for the determination of each element of this set it is sufficient to formulate an appropriate rule. The crossover and mutation operations of GA are similar to the original genetic operations. Therefore, the product of the GA operations can be called "descendant" or "child".

Let us now find the optimal arbitrary problem solution. Suppose there is a one-to-one correspondence between a string of characters and some of the desired solutions of the problem (it is a binary string in GA). Any solution can be encoded as a string and any string can be considered as a solution. By analogy with wildlife these strings can be called *genes*.

The next step is to assign a quality factor to each gene. This means that there is a criterion for choosing the most appropriate problem solution. Then the process of evolution consisting of two major operations on genes crossover and mutation begins. The crossover of two genes is to produce a new gene by gluing pieces of parental genes (see fig. 1)

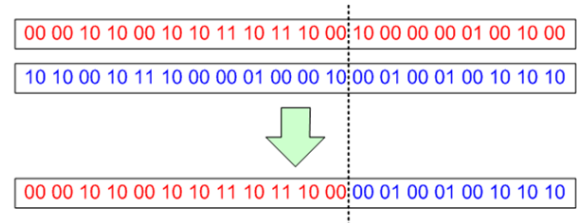


Figure 1. Scheme of crossover between two genes

The "child" partly inherits the properties of parental genes in terms of fitness both negative and positive ones. The replacement of one or more characteristics in a gene at random or by inverting the bit will be called gene mutation (see Fig. 2). As a result of a gene mutation an entirely new problem solution can be obtained (either better or worse than the initial one).

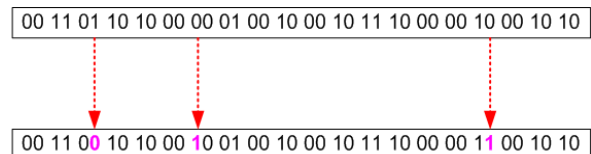


Figure 2. Scheme of gene mutation

After the crossover and mutation are carried out a new generation is re-evaluated and then a new stage of evolution starts. The optimal problem solution here includes the following points:

1. Forming the initial population, i.e. filling a random string array. The size of the initial population is one of the parameters and can vary depending on the problem.
2. Assigning some quality factor to each individual of a given population.
3. Sorting the population in descending order of a quality factor.
4. If the first gene of the population (the best solution) satisfies the problem conditions the evolution stops. Otherwise, crossing the first quarter of population to the second quarter and applying the mutation operator to the second half of population is needed [5].
5. Obtaining a new generation of solutions that are used when going to step two.

All stated above can be summarized in the following aggregative algorithm:

Step 1. Select some random sample $P_0 = \{\tilde{T}_i, \tilde{T}_j, \tilde{T}_k, \dots, \tilde{T}_z\}$ from set \tilde{T} . The size of the sample is not fixed and for each specific problem it can be set experimentally. This sample will be zero population in the GA evolution and selected elements of set \tilde{T} are individuals of zero population. Make zero population as current population.

Step 2. Sort out all individuals of the current population in descending order of individual quality q_i .

Step 3. If the best individual of the current population satisfies all the conditions of set C this individual is a problem solution and we quit. Otherwise the next Step is executed.

Step 4. Allocate a certain percentage of the best individuals and build the next generation in the population according to the following rules: the next generation consists of the best individuals, their children and mutated bad individuals (mutants).

Step 5. Consider a new generation as a current generation and go to Step 2.

Finally, the output will contain the solution that satisfies all the conditions of set C .

NURBS is one of the most employed surface fitting models, provided that it is a standard representation of curves and surfaces and is widely supported by modern standards like OpenGL and IGES, which are used for graphics and geometric data exchange. In addition, the NURBS surface model has stability, flexibility, local modification properties and is robust to noise.

The NURBS surface fitting problem is usually considered as a non-linear optimization problem. In order to find a good NURBS model from a large number of data, generally the knots, control points and weights are respected as variables [6]. In [7] binary-coded Genetic Algorithm is used for control point optimization and then knot values optimization and the error minimization of parametric surfaces as a global optimization problem is shown. In a similar way using GA, in [8], [9], optimization of both the knots and the weights corresponding to the control points for curve and surface fitting is done. In this study we consider a cubic NURBS surface dependent on control points and corresponding weights.

When using GA for the surface reconstruction on the basis of the point cloud by NUSBS it is necessary to define the basic terms of GA: gene, quality criterion, crossover and mutation. Let there be some surface found by the array of control points and corresponding weights. The explanation in this case is acceptable without loss of generality on the basis of NURBS-curve (see Fig.3).

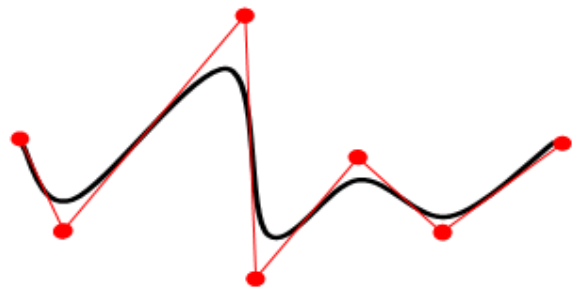


Figure 3. An arbitrary NURBS-curve

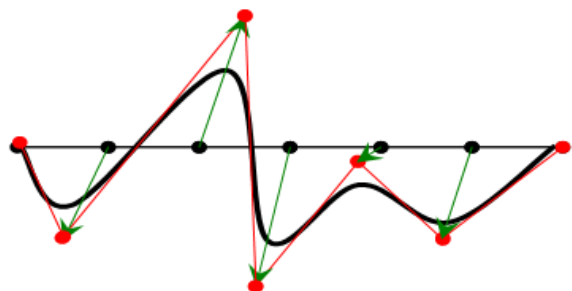


Figure 4. The vector defining the genome curve

We assign the array of co-ordinates of control points and weights to a surface on which each control point and weight is a vector corresponding to the deviation from the reference point at a fixed zero position (see Fig. 4). This can help to establish a correspondence between any of such surfaces and line vectors that can be called a genome. In this study weights are set to one to make NURBS uniform.

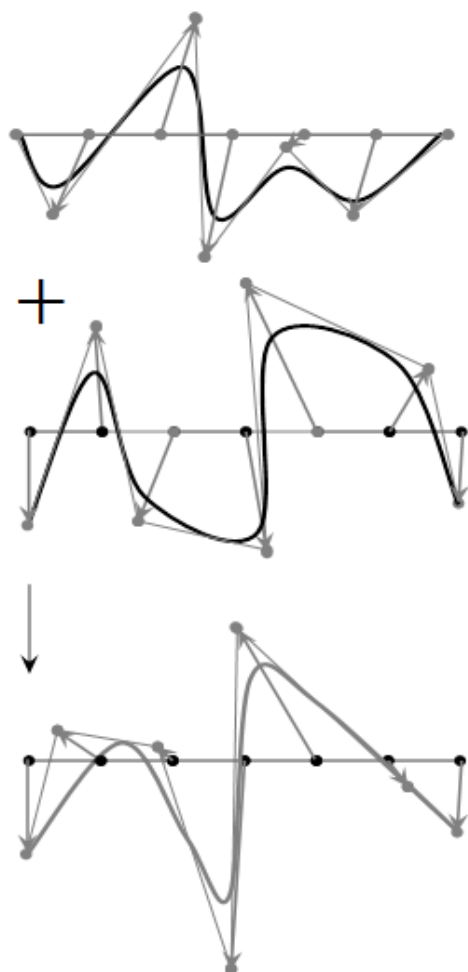


Figure 5. The crossover operation

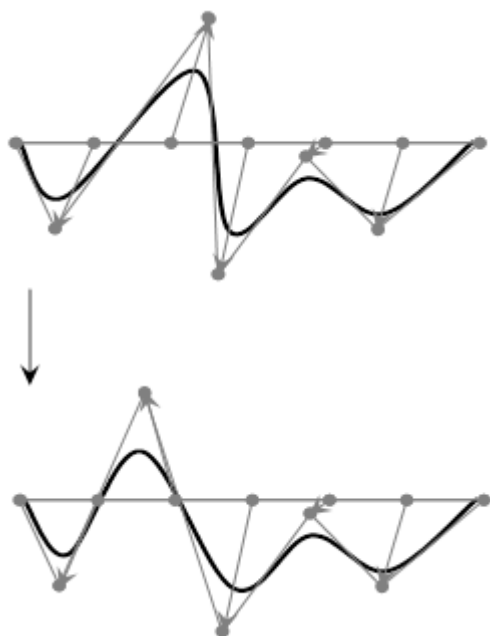


Figure 6. The mutation operation

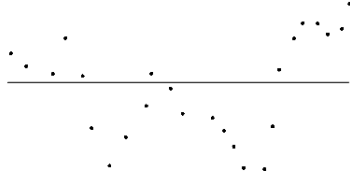
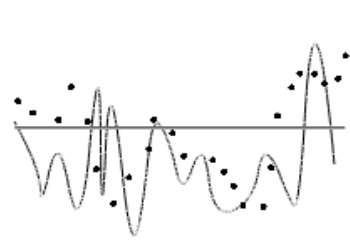
Next, let us introduce the concept of the gene quality. The criterion can be developed in such a way that the surface reconstructed from the point cloud is as close to a given point as possible. Therefore, the criterion is the minimal total (sum) distance between the point cloud and the NURBS. The smaller is the distance, the better corresponding surface satisfies the problem solution.

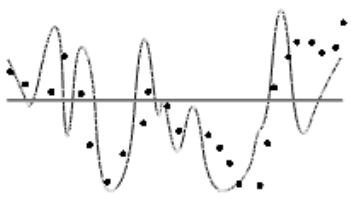
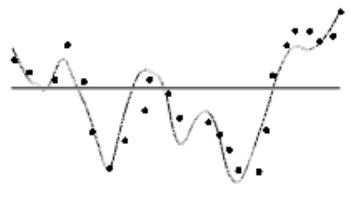
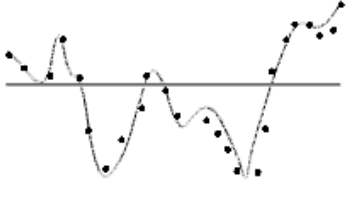
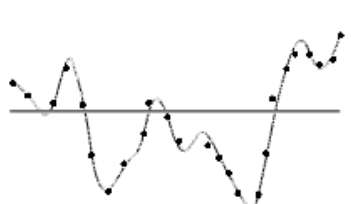
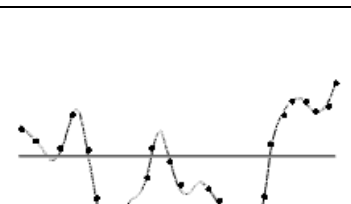
The gene of a descendant is formed on the parent genes basis by adding deviation vectors of the surface control points. This defines the crossover operation (Fig. 5). The mutation operation is done in the following way: any deviation vector in the mutating gene is replaced by a random vector (see Fig. 6).

The developed algorithm is tested for the fitting of the NURBS curve to the set of points. The process of fitting is shown in Table 1. The quality factor in Table 1 means the dimensionless sum distance between the point cloud and the NURBS curve normalized by the point cloud length at a horizontal axis. After 30 generations the evolutionary strategy reached the minimum, i.e. the quality factor decreased 4895 times in comparison with the non-optimized NURBS curve. The quality factor reached the value at about 0.12% of overall point cloud length that complies with requirements.

The best NURBS

Table 1

Generation #	Best NURBS	Quality factor
0		---
1		5.8746

5		3.5214
15		1.2341
20		0.3294
25		0.1546
30		0.0012

3. THE RETRIVAL OF A HUMAN FACE

In Fig. 7 the result of the retrieval process of a human face surface is presented. It is based on the points cloud obtained by 3D laser scanning. The best representatives of some

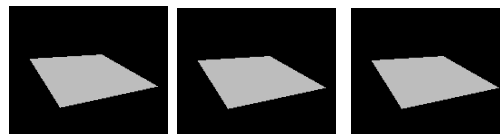
generations are shown in Fig. 7 arranged in triserial order. The criterion to choose the representatives is the minimum of total distance from all the points to the surface.

As we can see in Fig. 7 the quality of the NURBS representation of a human face rises from generation to generation. However, the best representatives at 1500th generation exhibit several artifacts (near the nose and the eyes, and at the boundary of the face). The artifacts at the boundary of the face can be easily eliminated by a trimming option. The artifacts inside the NURBS surface can be also eliminated by further steps of evolution. Generally the quality of the resulting surfaces can be very high. Besides, there are a lot of GA children in the evolution but it is possible to select the most appropriate for use depending on the goals.

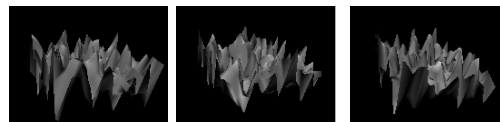
Initial set of points:



0- generation



5th generation:



900th generation:



1500th generation:



Figure 7. The best human face representatives of some generations arranged in triserial order

Thus, this approach allows us to model relatively complex objects such as a whole human face only by one NURBS-surface. A multivariate solution that is especially important in design can be considered as a

positive result. The example described above plays rather a testing role aimed at developing the approach.

4. THE SURFACE OF THE SHIP HULL

The use of GA is also proposed as a method for improving ship hull design through more effective exploration of the design space. The problem of creating fair ship hull surface is of major importance in Computer Aided Ship Design environment. The fairness of these surfaces is one of the most important conditions for the ship structure accuracy. It results in time saving and in improvement of the assembly and hull welding quality. It also reduces the cost of the structure significantly [10], [11], [12]. Currently several methods to control ship surface quality are used:

1. The Gaussian curvature visualization which allows identifying the problematic areas of the ship hull surface.
2. Visualization of the curvature radii of curves, surfaces and sections.
3. Visualization of the inflection points and inflection lines of sections and surfaces.
4. Dynamic change of the inflection lines and lines of curvature by the manual surface editing.
5. Visualization of surfaces and sections in a compressed form at one of the coordinate axes. It is a very important feature for modeling surfaces strongly elongated at one of the coordinate axes (wings, ruder etc.).
6. Automatic control of the surface deviation from the original data.

The described means of the surface quality control allows us to abandon the paper drawings completely, to reduce the simulation time sufficiently and to increase the quality of the simulated surface.

When designing the ship hull surface by means of shipbuilding CAD-systems the main purpose is to reduce manual labor. The advantages of GA use in this case are the following. First, genetic algorithm (GA) is a highly effective tool for the exploration of large-scale, nonlinear design spaces and, when combined with gradient based search techniques, may provide a more computationally efficient means of identifying near optimal designs. Second, GA method may provide a high utility tool that can enhance the ship design process. Third, the current design choice method of weighted objective measures of effectiveness can mask potentially useful areas of the design space. Therefore, the approach for the NURBS-surfaces retrieval on the boundaries of parametric quadrilateral defined by diametric buttock has been developed. The approach is based on GA and permits half of the hull surface to be designed without subdividing it into separate patches. Besides,

the design can be based on the general hull curves as well as on a point cloud that can be obtained by different ways including 3D scanning.

NURBS-surface retrieval on points cloud obtained on the basis of the main shipbuilding curves (buttocks, frames and waterlines, see Fig. 8) is carried out by means of the genetic algorithm with the scheme described in Section 2. Usually the main shipbuilding curves contain the information not only about the surface form, but also the information about the smoothness at key hull points and derivatives of any order. When forming a set of points on the basis of these curves all this information is lost. It means that the locus of points remains in a space only through which the surface should pass under design.

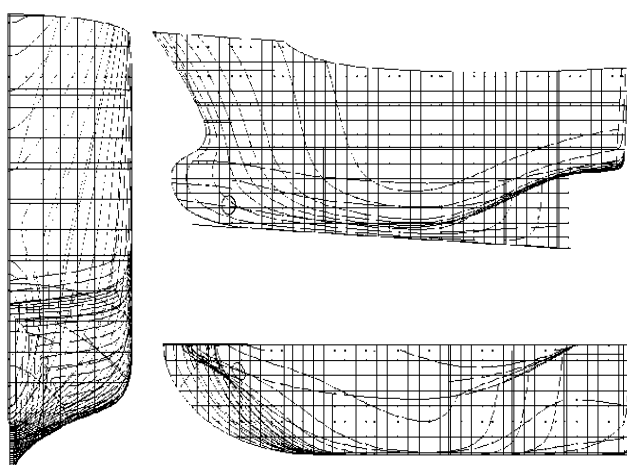


Figure 8. The theoretical drawing of seagoing trawler hull

The smoothness of the final NURBS-surface is determined by the degree of consistency of the source of the original theoretical drawing and smoothness of the main hull lines. An example of such points set to build a final NURBS-surface is presented in Fig.9.

Let us consider the process of the surface retrieval for the seagoing trawler as an example. The theoretical drawing of the trawler is shown in Fig.8. [13] The initial data in the form of a point cloud formed on the basis of hull curves are shown in Fig.9. Given that the smoothness of the surface depends on the smoothness of the source hull lines the latter have been smoothed by the Gauss algorithm. This allows obtaining the consistency with the main hull lines.

The Genetic Algorithm was used from the starting point for the half portion of the ship hull design space as described in Section 2. For the initial population we choose a set of NURBS-surfaces with the size of

the box bounding set of points and random deviations of nodes at the Y-axis. Default settings utilized initially included floating point genes. Crossover operations utilized simple procedure, i.e. genes were added between parents. All crossover rates are defined as the number of crossovers, regardless of the total population. Mutation was performed on a specified number of randomly selected individuals at each step of evolution. The quality of each individual surface was the sum distance from each point of cloud to the surface. Thus, we can obtain a set of populations at each step of evolution by standard GA operations such as mutation and crossover. In other words this set is a set of NURBS-surfaces that already meet the requirements of smoothness since they are based on the point cloud that is obtained in turns from the main shipbuilding curves. Finally, the GA allows us to get the surface closer and closer to the initial point cloud that is actually an equivalent to the "growing" of the required surface similar to a living organism.

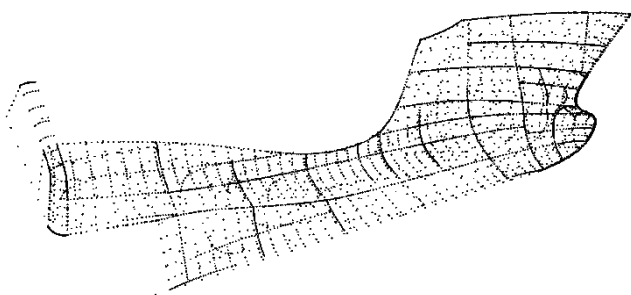


Figure 9. The initial point cloud to build NURBS-surface

The best representatives of the hull surface in some generations are shown in Table 2. The 6th method of ship surface quality control mentioned above is used, that is the automatic control of the surface deviation from the original data. The time of obtaining of each generation representative and the value of the sum distance from the initial cloud to the surface are presented in Table 3. This distance is used as a criterion for the quality of the surface retrieval. The improvement of the last individuals of subsequent generations did not occur.

From Table 3 it is clear that to obtain a ship hull surface represented by a single NURBS-surface with acceptable quality about 3000 generations are needed. This process takes approximately 1 minute of CPU (AMD Phenom™ II N930 Quad-Core Processor 2.00 GHz).

The best representatives of some generations

Table 2

generation #	Best individual
100	
500	
1500	
3000	

In this case, the overall deviation of the final NURBS-surfaces from the original point cloud does not exceed 83 mm. This corresponds to the shipbuilding accuracy of individual point deviation

from the surface up to $0.0001B$, where B is the width of the hull.

The quality factor

Table 3

generation #	Quality of the best individual (total distance), mm	Compute time, s
100	17254.612	4.34
500	12703.928	20.53
1500	9870.289	32.92
3000	82.731	48.93

Thus, on the basis of the mathematical NURBS approximation apparatus and the approach based on genetic algorithm for the retrieval of smooth complex ship hull surface is developed. The developed approach can be successfully used for the retrieval of this type of surfaces for which smoothness is a major requirement.

The final NURBS-surface of the trawler hull without plane part of the board is shown in Figure 10. The whole final NURBS-surface of the trawler hull is presented in Figure 11.

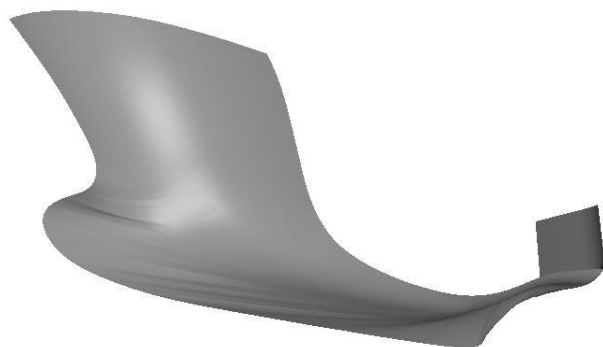


Figure 10. The final NURBS-surface of the hull without plane part of the board

4. CONCLUSION

In conclusion it should be stated that the NURBS-surface can be obtained by GA on the controlling points which may be solved on the given scattered point cloud. The genetic gene and evaluating function are given and the controlling points and

their corresponding weight genes are computed. By numerical simulation this approach is verified for the validity of the simplified representation of the fitting surface.

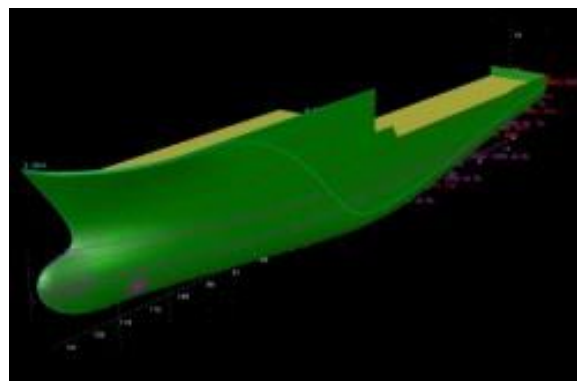


Figure 11. The whole final NURBS-surface

5. REFERENCES

- [1] Tomassini M. A survey of genetic algorithms. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, volume III, pages 87-118. World Scientific, 1995.
- [2] Kirkpatrick S., Toulouse G. Configuration space analysis of traveling salesman problems. *J. Phys. (Paris)* v46, 1985
- [3] Holland J. H. *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press. 1975.
- [4] Boyer, D. O., Martnez, C. H. & Pedrajas, N. G. Crossover Operator for Evolutionary Algorithms Based on Population Features, *Applied Soft Computing J.*, 39(3), 2007.
- [5] Aggarwal C. C., Orlin J. B., Tai R. P. Optimized crossover for maximum independent set. *Oper. Res.* v45, 1997.
- [6] E. Ulker, NURBS curve fitting using artificial immune system, *Int. J of Innovative Computing, Information and Control*, vol.8, no.4, April 2012.
- [7] A. Limaiem, A. Nassef and H. A. Elmaghraby, Data fitting using dual kriging and genetic algorithms, *CIRP Annals*, vol.45, pp.129-134, 1996.
- [8] F. Yoshimoto, M. Moriyama and T. Harada, Automatic knot placement by a genetic algorithm for data fitting with a spline, *Proc. of the International Conference on Shape Modeling and Applications*, pp.162-169, 1999.

- [9] M. Sarfraz, Representing shapes by fitting data using an evolutionary approach, *Int. J. of Computer-Aided Design & Applications*, vol.1, no.1-4, pp.179-186, 2004.
- [10] A. Swee Wen, S. M. H. Shamsuddin, Y. Samian., Ship Hull Fairing Using Nurbs, *Proceedings of the Postgraduate Annual Research Seminar*, pp.225-228, 2005.
- [11] A. Swee Wen, Optimization of ship hull NURBS surface fitting using simulated annealing, *University of Malaysia*, 2007.
- [12] K.G. Pigounakis, P.D. Kaklis and A.D. Papanikolaou: Ship Hull Fairing under Shape and Integral Constraints, *Proceedings of IMAM '95 / 5th Congress of the International Maritime Association of Mediterranean*, Dubrovnik, 1995.
- [13] Popov, E. V., Rekshinsky, A. V. A ship hull surface design using genetic algorithms, “*Vestnik IzhGTU*”, Periodic scientific and theoretical J., Izhevsk, # 3, 2007.

An Optimization of Square Parameterization

Anuwat Dechvijankit Hiroshi Nagahashi Kota Aoki
 Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology.
 4259 Nagatsuta-cho, Midori-ku, Yokohama-shi,
 Kanagawa, Japan, 226-8503
 dechvijankit.a.aa@m.titech.ac.jp longb@isl.titech.ac.jp aoki.k.af@m.titech.ac.jp

ABSTRACT

In order to parameterize a three-dimensional surface into a two-dimensional planar domain, we need to convert its polygonal mesh into a disk topology surface. For quality of texturing or re-meshing that uses a parameterization technique, it is more effective if the distortion of two-dimensional manifold planar domain map is as small as possible. Since square planar domain is easily understandable to human or very simple as a computer image file, it has been frequently used in real world applications. We introduce a series of experiments focusing on how to deliver an optimized square parameterization with low-cost calculation and stable result. The result of these experiments shows that our method is a suitable method for optimizing square parameterization.

Keywords

Mesh Parameterization, Optimization, Particle Swarm Optimization, Sampling

1 INTRODUCTION

Mesh parameterization is defined as a mapping between a 3D manifold surface and a suitable target domain. In general, the mesh parameterization is formulated as a mapping from 3D triangulated surfaces to a certain 2D planar domain. However, it requires the surfaces to be topologically equivalent to a disk without any hole. Parameterization between two domains generally causes distortion errors such as a stretch. Hence, low stretching is an important criterion for parameterization.

Unlike a circular boundary domain or a natural boundary domain, the square boundary domain has a specific characteristic that requires user-defined algorithms in boundary-mapping assignment (constraint part). Different positions in square boundary mapping can generate different quality of parameterization result as shown in figure 1.

The easiest way for delivering the lowest stretching square parameterization is to check all possible boundary mappings (brute-force) with a stretch-minimizing parameterization method. It can guarantee the best result, however the main problem of this approach is extensively time-consuming.

We did a series of experiments getting the lowest distortion square parameterization by avoiding the

brute-force high-computation method. The main goal of this paper is to devise an algorithm to deliver an optimal square parameterization from any kind of stretch-minimizing methods with fast and stable result. Moreover, since GPU-Computing has been introduced and widely been used nowadays, the devised algorithm should support parallel-computing scheme as well.

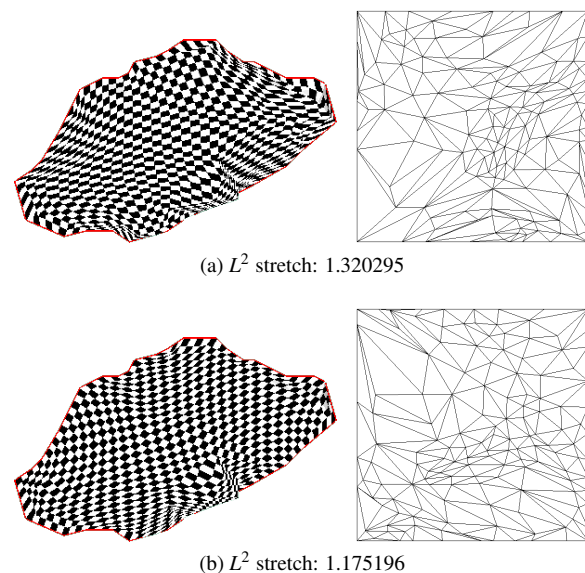


Figure 1: Ustica model with check-board texture mapping using same stretch-minimizing square parameterization with different boundary mappings. (a) shows the worst case that have largest L^2 stretch. (b) shows the best case that have smallest L^2 stretch. We can notice the different quality of textures around boundary area.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2 RELATED WORKS

Since texture mapping was introduced into computer graphics world by E. Catmull [Coo87a], it became a trend that every graphics board or graphics API must support it nowadays. Mapping a 2D texture onto a 3D surface requires some kind of parameterization of the surface. The parameterization result can be represented by planar coordinates u and v , indicating a position in 2D domain related to a position (x,y,z) in 3D domain.

Many well-known parameterization methods have been proposed. Tutte[Tut63a] used a barycentric mapping theory and created a conformal mapping. Floater[Flo97a] used relative angles as a weight in each interior vertex to create barycentric mapping. Later on, stretch-minimizing methods have been proposed to achieve low stretch as possible. Sander et al. [San01a] used geometric-stretch matrix as the sum of squared singular values, and minimized it as a non-linear system. Yoshizawa[Yos04a] proposed a fast method of stretch-minimizing by recomputing the weight of linear energy-minimizing equations by using previous stretch value as a divisor.

Concerning the limitation of fixed-boundary parameterization, Least Squares Conformal Maps (LSCM)[Lev02a] were presented as alternative ways to optimize the boundary positions from fixed-boundary into free-boundary. They used different harmonic energy formulations found in harmonic map[Eck95a] but still minimized angular distortion. Intrinsic parameterizations[Des02a] used the same technique found in LSCM to preserve angle distortion, and preserved area distortion. Both of them could significantly improve the distortions. However, they aimed to optimize by changing fixed-boundary into free-boundary parameterization, not square-boundary one.

To guarantee the generation of a valid parameterization without local or global fold overs and the control of each mesh triangle distortion to not exceed a certain threshold, Sorkine[Sor02a] proposed bounded-distortion concept with simultaneously seam-cutting. Lipman[Lip12a] also proposed bounded-distortion mapping spaces which can control worst-case conformal distortion, orientation preserving and one-one mapping in various existing mapping algorithms. However, they aimed to control mappings at unconstrained part.

3 SQUARE BOUNDARY MAPPING

Generally, planar parameterization requires some constraint values in its solving system. For doing a fixed-boundary parameterization, it requires a user-defined boundary position mapping as constraint values before solving interior coordination. Unlike circular parameterization that averages each boundary edge length and

circle angle, square parameterization needs some user-defined algorithms in the assignment of boundary mapping. There are no specific algorithms for it. Users can create their own algorithm based on the length or numbers of boundary edges and so on.

In our square boundary mapping algorithm, a mesh M has n boundary vertices. Let boundary edges be $E_B = ((v_{b1}, v_{b2}), (v_{b2}, v_{b3}), \dots, (v_{bn}, v_{b1}))$, having total length d . Let a list of all boundary vertices be $V_B = (v_{b1}, v_{b2}, \dots, v_{bn})$ and it is sorted in the order of connection of E_B . We call v_{b1} as a reference start point of V_B and E_B . Let $P_{i,j}$ be a corner position in square planar domain as shown in figure 2 and u_s be the length of each side of square planar. We try to map some edges in E_B onto a side of square planar. In order to achieve lowest stretch at boundary area, one side should be mapped by a quarter of E_B based on the total length of edges ($0.25d$).

Let v_b be a vertex in V_B that we want to map onto $P_{0,0}$. Let the distance from v_b to v_{b+m} be l . We try to find v_{b+m} whose distance l is equal $0.25d$. However, in most cases it is not equal. Therefore, we find v_{b+m} that has $l \approx 0.25d$. Then, we map $(v_b, v_{b+1}, \dots, v_{b+m})$ whose total length is l onto the square side from $P_{0,0}$ to $P_{1,0}$ relatively on each edge length over u_s .

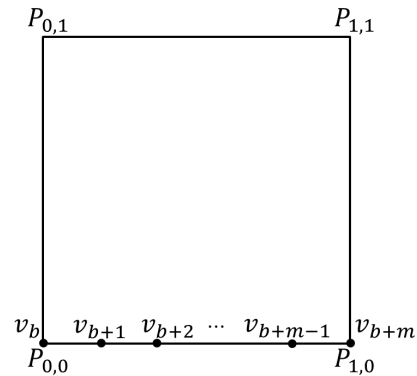


Figure 2: shows a mapping sequence on planar points from $P_{0,0}$ to $P_{1,0}$.

As for other sides of the square, we can follow the same process described above by locating v_{b+m} to the corner $P_{1,0}$ and iterate the mapping process to $P_{1,1}$, $P_{0,1}$ and finally back to $P_{0,0}$.

4 OPTIMIZATION

Square parameterization has a unique characteristic that requires a user-intervening boundary position mapping. The different boundary positions in planar domain can give moderate margin of stretch (see figure 1). The problem is how we can obtain the optimal square parameterization.

Considering our problem of square boundary optimization, we map the boundary vertices in mesh domain

onto the boundary in square domain with some conditions. One important condition is the relationship between a side of square boundary and mapping boundary edges. Our mapping method is trying to map a quarter of total boundary edges in terms of length onto one side of the square that we mentioned in section 3. With this condition, a new complexity arises. That is, each boundary mapping could have different number of edges on each side of the square. It is impossible to incorporate these boundary conditions into a linear solving system.

The simplest way is a brute-force approach using a stretch-minimizing parameterization. In our mapping method, brute-force means that we let every vertex in V_B be mapped onto $P_{0,0}$ then do stretch-minimizing parameterization. Although it guarantees the best answer, one time stretch-minimizing parameterization on a fine mesh might consume not a few amount of time. Although, we can speed up by applying parallel-processing but doing brute-force and checking every possible boundary mapping on the square boundary might consume a lot of time.

25 percent of brute-force

Let the first boundary mapping be v_{b1} onto $P_{0,0}$, $v_{b\sigma}$ onto $P_{1,0}$ and $v_{b\tau}$ onto $P_{1,1}$ ($v_{b1}, v_{b\sigma}, v_{b\tau} \in V_B$) that are assigned by our boundary mapping algorithm. It means the distance from v_{b1} to $v_{b\sigma}$ should be approximately $0.25d$, also the same for distance from $v_{b\sigma}$ to $v_{b\tau}$.

By starting from v_{b1} , we sequentially assign a vertex v_b onto $P_{0,0}$, and map the following vertices onto the square domain $[P_{0,0}, P_{1,0}]$. After repeating the mapping for interval of a quarter of boundary edges, then the vertices $v_{b\sigma}$ and $v_{b\tau}$ might be mapped onto $P_{0,0}$ and $P_{1,0}$ respectively. It is the same as we rotate the first mapping (v_{b1} onto $P_{0,0}$) 90 degree as shown in figure 3.

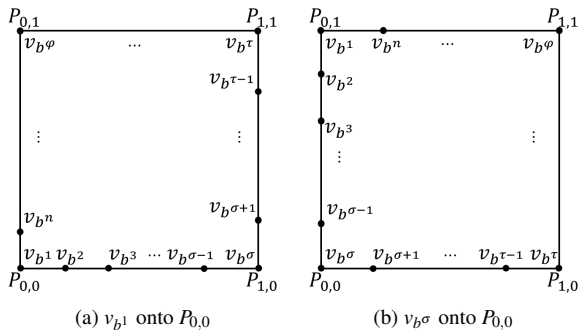


Figure 3: shows a similarity of boundary mapping after shifting for a quarter of total length of boundary edges.

From this property, we can reduce the number of testing cases to around 25 percent because we can rotate the parameterized planar from a boundary mapping ($v_{b1}, v_{b2}, \dots, v_{b\sigma}$) to obtain the result of the rest

mapping ($v_{b\sigma+1}, v_{b\sigma+2}, \dots$). However, doing brute-force with around 25 percent of total testing cases still consumes much of time. Our goal is finding the optimal square parameterization while keeping the calculation time less than 25 percent of brute-force.

4.1 Faster parameterization methods

We examined the time consuming issue of the brute-force approach. We could notice that stretch-minimizing parameterization is a main reason of the problem. Since it is iterative process and it seeks converging of the energy or stretch, its calculation time is much more than the time of one parameterization by means of linear solving system. If we can replace a stretch-minimizing to a faster solving parameterization in the optimization process, then it should reduce the consuming time a lot. Here, we set our hypothesis that every parameterization method should give a same direction result; the best boundary mapping by fast-solving method is same as the boundary mapping by stretch-minimizing method. Now, the concept "same direction" is defined as follow:

Same Direction

We assume that there are two square boundary mappings; M_n and M_{n+1} . In addition, we have two parameterization methods F and G . Let R_n and R_{n+1} be parameterization results if we do F on both M_n and M_{n+1} . Moreover, S_n and S_{n+1} are parameterization results if we do G on both M_n and M_{n+1} . If R_n is better than R_{n+1} and S_n is better than S_{n+1} then we say that F and G have a same direction.

4.1.1 Experiment and Result

We setup the experiment to test the hypothesis. One parameterization is fast solving but high potential of stretch. The other is slow solving but stretch-minimizing. We tried to use fast solving methods to predict which boundary mapping is the best for square parameterization. We did the experiment to observe the stretch of unit square boundary by various fast solving parameterization methods of Shape-Preserving[Flo97a], Tutte[Tut63a], mean-value[Flo03a] and harmonic map[Eck95a]. We compare L^2 stretch[San01a] value from our candidate methods with value from stretch-minimizing square parameterization using [Yos04a].

As a result, we could not find any relationship among them. When observing the lowest stretch of all methods with same boundary positions, they do not give the lowest stretch in the same direction. Stretch observation from the fast solving method does not enable to check which mapping boundary points is optimal setting for stretch-minimizing method as the hypothesis.

We concluded that optimizing square parameterization needs to be done and analyzed directly from the stretch-minimizing method.

4.2 Heuristics

From previous experimental result, we cannot use a fast solving parameterization. Instead, we need to use stretch-minimizing one that requires a lot of calculation. For that reason, we set our approach reducing the number of calculations. If we can pursue the optimization by doing parameterization with few testing cases, then it will surely be better than brute-force method.

We attempted to find a best optimization among the existing ones. There are many techniques and they are divided into 3 categories[Wik13a]. Optimization algorithms such as a Simplex algorithm are suit for linear or quadratic programming solving. Iterative methods such as Newton and Quasi-Newton methods suit for non-linear programming solving. Heuristic algorithms such as Genetic algorithms or Hill climbing suit for solving the problems that cannot be solved or too slow by classic methods.

When we consider our mentioned problem, we concluded that a heuristic algorithm may be the best one for boundary optimization problem. The reason is that our boundary optimization problem has the difficulty of two sub-problems connected together. One sub-problem is concerned with our main problem, i.e., finding best mapping of boundary vertices and edges. The other is a parameterization problem of finding planar location of interior vertex. It is too difficult to combine the two sub-problems into one problem solving system since it has unique condition for the boundary mapping. In addition, we try to find global optimum of our problem, not local ones. To find one local optimum does not show or know it is globally optimal until all local optimums are found. From these reasons, well-known algorithms such as Simplex or Newton do not fit to our problem. Hence we chose a heuristic method to solve our optimization problem.

4.2.1 Particle Swarm Optimization

We choose "Particle Swarm Optimization" algorithm (PSO) [Ken95a] for solving our optimization problem. It is a new swarm intelligent technique, originally inspired by social behavior of animal flocking. PSO has been used mainly to solve unconstrained, single-objective optimization problems. The advantage of using PSO is that it does not use the gradient of the problem to be optimized, so the method can be readily employed for optimization problems. This is especially useful when the gradient is too laborious or even impossible to derive. This versatility comes at a price, as PSO does not always work well and may need tuning of its behavioral parameters so as to perform well on the problem at hand[Eri10a]. It requires 3 parameters of effective factor from velocity, local-best position and global-best position.

Next velocity of each particle is updated based on current velocity, local-best and global-best positions of the

particle. Moreover, it is received effectively from these 3 parameters and random numbers. Then, each particle's next position is calculated by using its new velocity. Let i be the number of particles, k be iteration times and r be a random number in searching scope. Then, we assume that a , b_l and b_g are user-defined effective factor from current velocity, local-best position and global-best position, related as follows respectively. In equation forms, they are

$$v_i^{k+1} = av_i^k + b_l r_1 (pl_i^k - x_i^k) + b_g r_2 (pg^k - x_i^k) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2)$$

Algorithm 1 summarizes a standard PSO algorithm.

```

\\initialize particle...
for each particle  $x_i$  do
     $x_i \leftarrow r$ 
     $pl_i \leftarrow \text{unknown}$ 
\\starting main iteration...
repeat
    for each particle  $x_i$  do
         $y_i^k = f(x_i^k)$ 
        if  $y_i^k$  better than  $f(pl_i^k)$  then
             $pl_i^k \leftarrow x_i$ 
         $pg^k \leftarrow \text{best of all } x_i^k$ 
        for each particle  $x_i$  do
             $v_i^{k+1} \leftarrow av_i^k + b_l r_1 (pl_i^k - x_i^k) + b_g r_2 (pg^k - x_i^k)$ 
             $x_i^{k+1} \leftarrow x_i^k + v_i^{k+1}$ 
    until  $k > \text{maximum iterations}$  or
         $pg$  unchanged many times
    optimum  $\leftarrow pg$ 

```

Algorithm 1: Pseudo-code of PSO algorithm

4.2.2 Experiment and Result

We did experiments of applying one dimensional PSO to our problem. Let a particle position x_i be the distance from the reference start point v_{b1} . First, search the nearest $v_{b\alpha}$ in V_B whose distance from v_{b1} is closest to x_i , and assign it to v_b that is mapped at lower-left corner $P_{0,0}$. Then we follow the algorithm described in section 3. At last, we do square stretch-minimizing parameterization using [Yos04a] method to obtain interior coordinates in 2D planar domain and then calculate L^2 stretch value for evaluation. On each mesh, different number of particles and various changing factors of local and global best positions were examined until the system gave optimal answer. We did 10 times per one parameters-setting and get statistic results since PSO algorithm is based on a random process as shown in equation 1.

As a result, we could reduce the calculation time to around 50 to 75 percent comparing to 25 percent of

brute-force approach. By changing user-defined parameters, we could improve calculation time in some mesh models. However, we could not gain an expected result from the turning of these user-defined parameters yet.

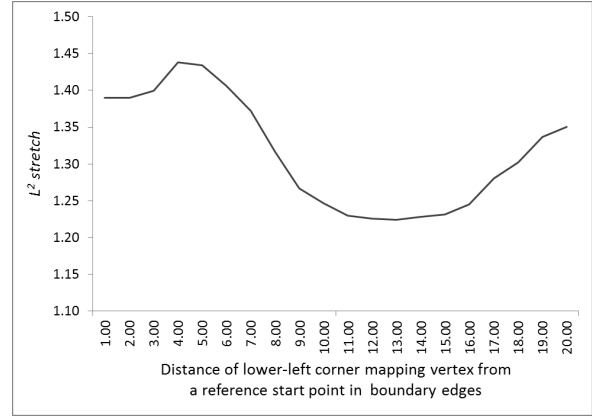
The number of particles plays important roles; more particles secure the best answer (same as brute-force's answer) but calculation cost increases. The algorithm itself is based on random process, and it may give unstable optimal answer if we use small number of particles. Increasing the number of particles will cost almost the same calculation time as 25 percent of brute-force approach. Moreover, PSO has a disadvantage point on parallel-computing because it updates particle positions based on global-best position at each iteration which prevents from doing large numbers of parallel-processing simultaneously.

We concluded that PSO can improve the performance when using an appropriate number of particles. Even, the performance was still not as we expect but its algorithm of checking few positions and focusing around seems to be appropriate with our optimization problem. The main bottleneck of PSO in our problem is a procedure of random search. Avoiding the random search while checking few positions should generate better performance stably.

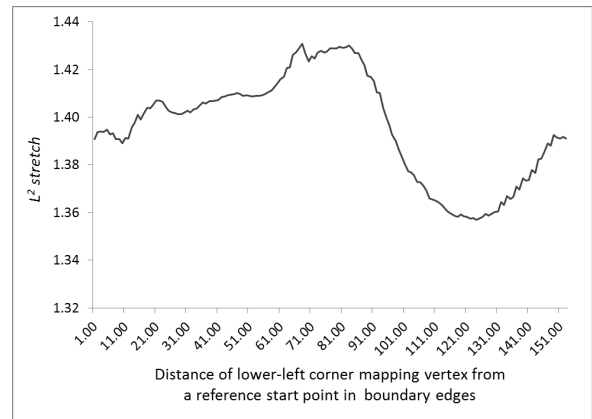
4.3 Sampling

The PSO algorithm is based on a sampling approach. Each particle acts as a sampling unit and every time a particle moves to a new position, it will examine that position. In our case, it means doing one parameterization. If we observe a particle movement, we can notice that the particle will move toward previous local-best and global-best positions. Therefore, initial global-best particle is important that affects to the performance and stability. Each particle's initialization can narrow down searching scope a lot if a position is located at optimum area because the global-best particle is one of local-best particles. We adapted these concepts into a sampling algorithm by narrowing down searching scope of the initial sampling.

We started to analyze square parameterization by looking at brute-force results. We noticed that most of our test models' stretches are changed gradually when we change v_b at $P_{0,0}$ along V_B (see figure 4). From this characteristic, we can reduce the number of calculations by focusing attention on the boundary-mappings that have high potential to give an optimal result. In order to be able to do such thing, we need to know where is appropriate searching scope. If we plot stretch values, we are searching for potential area having global optimum. In other words, it is important to find a list of mapping $(v_{b-p}, \dots, v_b, \dots, v_{b+q})$ at $P_{0,0}$ that generates an optimal square parameterization.



(a) hand model



(b) Stanford bunny model

Figure 4: The graphs that show stretch values from doing square stretch-minimizing parameterizations (25 percent of brute-force) on testing models.

The problem is how to determine the searching scope. We use a sampling approach as a survey of stretch values same as PSO. Sampling is the reduction of a signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal). In our problem terms, we reduce the number of parameterizations to few cases so we can determine our searching scope. We do not use complex algorithms like pattern-search or random-search but we do a static sampling.

4.3.1 Step-Sampling

We propose a simple algorithm named step-sampling. It samples stretch values from selected boundary mappings. They will be selected as a step defined by user. After sampling was completed, we will get a boundary mapping that gives the lowest stretch as a center of optimal area. At last, we do deep-checking in that area. We check its neighbors that are still be unchecked for stretch values. The test case (boundary mapping) that gives the lowest stretch is an optimal answer.

Algorithm 2 summarizes our step-sampling algorithm.

Model	Total Test Cases	step											
		2	3	4	5	6	7	8	9	10	11	12	13
Hand	18	61.11%	44.44%	44.44%	44.44%	44.44%	50.00%	55.56%	55.56%	61.11%	66.67%	72.22%	77.78%
Head	20	60.00%	55.00%	55.00%	60.00%	70.00%	70.00%	85.00%	95.00%	100.00%	100.00%	100.00%	100.00%
Bunny Holes	153	51.63%	35.95%	29.41%	25.49%	23.53%	22.22%	22.22%	21.57%	22.22%	22.22%	22.88%	23.53%
Bunny No hole	123	52.03%	36.59%	30.08%	23.58%	21.14%	19.51%	18.70%	17.89%	17.89%	17.89%	17.89%	17.89%
Max	112	51.79%	37.50%	30.36%	27.68%	25.89%	25.00%	25.00%	25.89%	26.79%	27.68%	28.57%	29.46%
Cow	240	50.83%	35.00%	27.50%	23.33%*	20.83%	19.58%*	18.33%	17.92%*	17.50%*	17.50%	17.50%	17.92%*

Table 1: sampling method result: symbol * indicates that optimal answer is not same as brute-force's result. Bold numbers mean the lowest percentage of calculation.

```

step ← 2, 3, ... \user defined step
m = number of vertices in first 0.25d of EB
stretch[] ← ∞ \array m size

\\initial step sampling
for i ← 1 to m do
  if i mod step = 1 then
    stretch[i] ← SquareParam(vbi as P0,0)

\\deep-checking
l ← index of stretch[] that has best result
J[] = [..., l-2, l-1, l+1, l+2, ...] where
  stretch[] = ∞ and close to l
for j ← each J do
  stretch[j] ← SquareParam(vbj as P0,0)
optimum ← index of stretch[] that has best result

```

Algorithm 2: Pseudo-code of step-sampling

4.3.2 Experiment and Result

We did experiments on various models with various step values. Let the total test cases (boundary mappings) be m that can be calculated from the number of vertices in the first 0.25d interval of E_B starting from v_{b_1} . This number can be considered as a calculation time of 25 percent of brute-force approach. In our experiment, we count the numbers of doing stretch-minimizing parameterization as calculation time, including both initial step sampling and deep-searching period. After all, we represent the number of calculation times as percentage of total test cases m . We also check the optimal answer to be same as brute-force or not. Table 1 shows our results.

The result shows that we could reduce percentage of the calculation times to around 17 to 60 percent of total test cases. The step value plays important roles; appropriate step value can reduce more than half of total test cases while keeping the best answer. As for a "cow" model, which was converted from originally genus 4 to genus 0 (disk topology mesh), we could not obtain appropriate disk topology mesh. That caused a strange fact that stretches are changed extremely as shown in figure 5.

Considering about parallel-computing, our step-sampling algorithm can perform well without dependency on each boundary mapping.

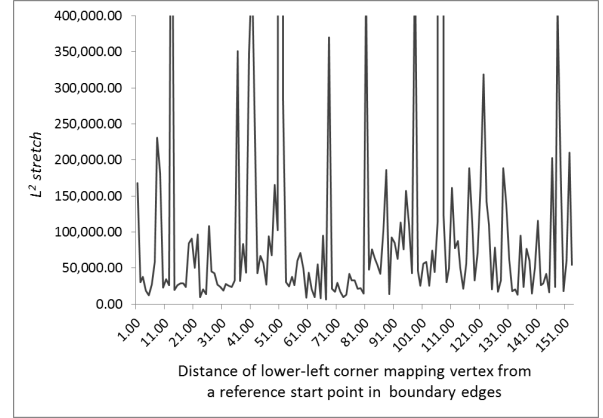


Figure 5: A graph that shows stretch values from doing square stretch-minimizing parameterization on a "cow" model that was converted from genus 4 to genus 0.

4.3.3 Step Value

From the result of step-sampling, proper step value can reduce calculation time a lot. However, too large step can result spending more calculation time than smaller one because larger step means larger searching scope.

We propose a formula for proper step value.

$$step \approx \sqrt{\frac{\text{number of vertices} \times \text{total test cases}}{\text{number of faces}}}$$

The step values from our proposed formula could reduce the calculation time to the lowest rate or nearly in most cases. Table 2 shows the results on our test models that be used same as table 1.

Model	Vertices	Faces	Total Test Cases	Step
Hand	1085	2006	18	3.12
Head	713	1357	20	3.24
Bunny Holes	36179	69463	153	8.93
Bunny No hole	35070	69644	123	7.87
Max	49342	98262	112	7.50
Cow	17135	33316	240	11.11

Table 2: Step value from our proposed formula on each testing model.

5 CONCLUSION

The easiest way for delivering the lowest stretching square parameterization is to do brute-force approach with a stretch-minimizing parameterization method. However, it will consume a lot of calculation time.

We have presented a novel approach to optimize square parameterization that enables to reduce calculation time a lot. From our experiments, optimizing square parameterization could not be considered from faster parameterization method; instead, it should directly do stretch-minimizing method to obtain optimal result. Also, we tried to apply an existing optimization; particle swarm optimization to our problem. It still could not reduce calculation time as expectation. However, we analyzed PSO algorithm itself then applied the concept of sampling to create our approach.

We propose our "step-sampling" concept which reduces much calculation time while maintaining a stable optimal result. Although it is a simple algorithm, we can have great performance that reduce more than half from brute-force approach. We also propose a formula to calculate suitable step number based on mesh's information that will minimize the calculation time.

We are still interested to improve the performance of our "step-sampling" approach. There are many procedures that might able to be improved such as deep-searching section or changing the way of doing static stepping.

6 ACKNOWLEDGMENTS

We would like to gratefully thank all reviewers, Shin Yoshizawa for C++ code of his stretch-minimizing parameterization [Yos04a], Hugues Hoppe for filled holes bunny and hand model data.

The models are courtesy of the Stanford University (bunny), the University of Washington (head), MPI für Informatik (max) and AIM@SHAPE (cow and ustica). This work was supported by JSPS KAKENHI Grant Number 24300035.

7 REFERENCES

- [Coo87a] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *SIGGRAPH Comput. Graphics*. 21, 4 (August 1987), 95-102., 1987.
- [Des02a] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Comput. Graphics Forum*, 21(3):209-218, 2002.
- [Eck95a] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH'95*, pages 173-182, 1995.
- [Eri10a] M. Erik and H. Pedersen. Good parameters for particle swarm optimization. Hvass Laboratories Technical Report, HL1001, 2010.
- [Flo97a] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231-250, April 1997.
- [Flo03a] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20:19-27, March 2003.
- [Ken95a] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942-1948 vol.4, nov/dec 1995.
- [Lev02a] B. Levy, S. Petitjean, N. Ray, and J. Mailliot. Least squares conformal maps for automatic texture atlas generation. In *ACM, editor, ACM SIGGRAPH conference proceedings*, Jul 2002.
- [Lip12a] Lipman Yaron. Bounded distortion mapping spaces for triangular meshes. *ACM Trans. Graph.* 31, 4, pages 108:1–108:13, July 2012.
- [San01a] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH'01*, pages 409-416, 2001.
- [Sor02a] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proceedings of the conference on Visualization '02. IEEE Computer Society*, pages 355-362, 2002.
- [Tut63a] W. T. Tutte. How to draw a graph. *Proceedings of The London Mathematical Society*, s3-13:743-767, 1963.
- [Wik13a] Wikipedia, Mathematical optimization, http://en.wikipedia.org/wiki/Mathematical_optimization, 2013.
- [Yos04a] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *SMI'04: Proceedings of the Shape Modeling International 2004*, pages 200-208, 2004.

Part-based Construction of digitized 3D objects

Daniela Borges
INESC-ID/IST/Technical
University of Lisbon
R. Alves Redol, 9, 1000-029
Lisboa, Portugal
daniela.borges@ist.utl.pt

Alfredo Ferreira
INESC-ID/IST/Technical
University of Lisbon
R. Alves Redol, 9, 1000-029
Lisboa, Portugal
alfredo.ferreira@inesc-id.pt

ABSTRACT

Nowadays, a few 3D acquisition devices are available at low-cost. While 3D capture is a commonplace, decompose the object into its components is not an easy task. Segmentation can help address this problem by supplying data which may be used to identify object components. However, it might not give complete and accurate information about components. In a context where a digital repository with every component that can belong to physical objects is available, retrieval algorithms can be used to construct a composed 3D model.

We propose a four phase solution to construct 3D digitized objects. We use Microsoft Kinect[®] to acquire 3D physical objects. A segmentation algorithm based on color information decomposes the object into a set of sub-parts. The component repository is queried using a shape-based retrieval algorithm, in order to identify which sub-part corresponds to each virtual component. Then, a 3D model of the physical object is constructed by assembling the retrieved components.

The work presented in this paper has a wide application domain, ranging from entertainment to health or mechanical industry. To validate our proposal, we implemented a toy-problem and evaluated its precision and efficiency. We used LEGO[®] blocks, which can provide challenges similar to real-world applications. The results were encouraging and we believe that our approach may even work better with greater object components, geometrically less similar to each other.

Keywords

Reconstruction, Acquisition, Segmentation, Retrieval.

1 INTRODUCTION

Technological advancements allowed the storage of objects such as audio, image and video through personal devices. These assets, previously considered tangible, can now be transported everywhere and are named digital media. Nowadays, 3D scanners are more accessible for anyone and several approaches have emerged to solve acquisition, analysis, classification, index and retrieval problems [VMC96]. Naturally, this also led to 3D model construction and new challenges have appeared.

Reconstruction techniques allow the acquisition of physical objects, in order to reach a digital model. However, if the object is composed by several components, it is impossible to identify every object component with a simple reconstruction.

Object construction is not a trivial challenge, because 3D object acquisition does not performs matching between known models and acquired objects. Moreover, the acquired objects could have several physical components which are also not identifiable. These considerations lead to an increase in the number of enthusiasts both from research and industry. Life of George¹ and Autodesk 123D² are examples of applications that had shown, respectively in 2D and 3D, the construction potential and why this is an interesting area. The establishment of new low-cost scanner devices increasingly accessible to anyone is also a motivation for our work.

Our research tries to ascertain if is possible to use a low-cost device to acquire a 3D object and perform a segmented construction that uses color and shape information. In other hand, we acquire, segment and retrieve a 3D physical object an all its components, in order to create a reliable 3D digital model.

To support our vision, we address two major challenges: (1) the segmentation of the acquired polygonal mesh, using color information; and (2) the identifica-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

¹ <http://george.lego.com/>

² <http://www.123dapp.com/>

tion of segmented physical components (sub-parts), that allows the construction. To be able to perform the construction, our work will assume two restrictions. The former is that adjacent components must have different colors and the latter is that every physical component allowed to be acquired must be known, i.e., must have a model (virtual component) in a repository. Therefore, our proposal aims to reconstruct 3D physical objects in real-time with an acceptable success rate. To accomplish these requirements, we selected algorithms that meet a tolerable time-quality relationship.

Next section discusses related work, presenting several approaches to solve construction problem through acquisition, segmentation and retrieval phases. Section 3 describes our solution and compares it with other approaches referred in the related work. Section 4 mentions how to evaluate our algorithm performance, shortcomings analysis and present relevant results. Finally, we present our conclusions and point out some future work.

2 RELATED WORK

3D object acquisition is allowing the scanning of a huge amount of objects mainly in the fields of computer-aided design (CAD), computer-aided manufacturing (CAM), cultural heritage, reverse engineering, among others. Range scanners, presented in [BR02], grant object acquisition and are divided in several systems such as triangulation systems, time-of-flight systems, among others. Although high range scanners (such as Comet L3d³ or EXAscan⁴) have huge accuracy and resolution, some low-cost scanners that appeared recently are increasingly widespread. As a result, some applications have emerged primarily in the area of video games.

Low-cost devices, such as Microsoft Kinect⁵ or Primesense sensor⁶, are becoming increasingly available to anyone, regarding the low price in comparison with other scanners. However, although these scanners acquire real-time RGBD data and produce relevant results through controlled scenarios, they do not have high accuracy. That is a high disparity between scanner prices is a reasonable reason for us to choose Microsoft Kinect[®] to acquire our 3D models. Moreover, we also consider the popularization around this scanner due to its low price. Because of this, lots of users are now using Kinect[®] for a wide variety of applications.

Segmentation is useful for location, classification and feature extraction of 3D shapes. Although segmentation is easily performed by humans, computers need complex algorithms to achieve the same work. There are several segmentation algorithms that receive a point cloud, an object or an image as input. The goal is to decompose the object into patches or regions whatever is the input received. Some clues may help to reach segmentation such as normal calculation, curvatures or concavity around the boundaries. Up to now, in the field of segmentation a large number of algorithms have been proposed such as Region Growing [AB94], K-means [STK02], Fitting Primitives [AFS06], among others. We need to take into account that some algorithms must be used offline whereas others are faster and consequently better to interactive applications.

Although the referred algorithms are traditional, new technologies bring depth and color information. Moreover, the combination of color and depth information to segment shapes is becoming common use.

One approach to construct a 3D model is to retrieve every component that belong to an object, identifying them. First of all, we need to index every possible component through the descriptor computation for every model. Descriptors define models through a signature and provide a way to retrieve those models efficiently. Therefore, have appeared several descriptors, namely Spherical Harmonics (SHA) [FMK⁺03], Light-Field Descriptor (LFD) [CTSO03], and more recently BOW-LSD [LSFG11] or PatchBOF [TDVC11]. Our work requires almost real-time and, consequently, we decided to use the most efficient descriptor. When all models are indexed, preferably using an indexing mechanism, we are allowed to retrieve them. Models are retrieved using queries that represent those models. This query is also the result of a descriptor computation. Afterwards, when the query is performed the most similar results may be retrieved.

A pioneer work in construction area is The Digital Michelangelo Project [LRG⁺00]. This project, which major requirement is the construction of high resolution digital models, uses specific software and hardware in order to scan cultural heritage. A triangulation system was used to acquire depth information, with the help of one motorized gantry. Due to the hardware used the final output contains billions of polygons (for instance, in David statue) which is not possible to handle with commercial applications. Our approach is different to this work because we use low resolution models, that have different requirements.

Recently, there are other projects [SW11, MMWG11] in the field of cultural heritage. Schwartz et al. [SW11] presented a work where 3D geometry is extremely required and optical properties of object surface are also desired (such as reflexion). Using a Bidirectional Tex-

³ <http://www.steinbichler.com/products/surface-scanning/3d-digitizing/comet-l3d.html>

⁴ <http://www.creaform3d.com/en/metrology-solutions/portable-3d-scanner-handyscan-3d>

⁵ <http://www.microsoft.com/en-us/kinectforwindows/>

⁶ <http://www.primesense.com/solutions/sensor/>

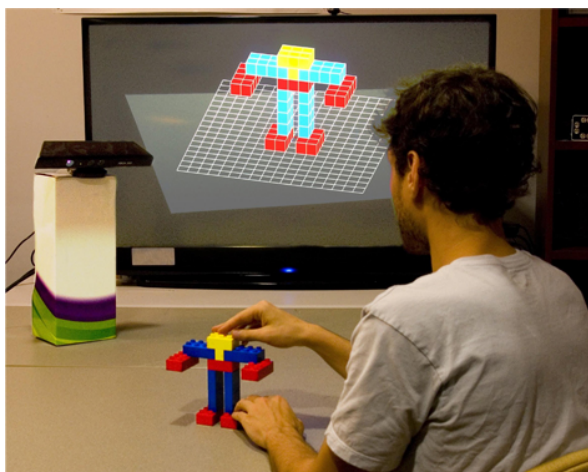


Figure 1: Setup of Lattice-First project.

ture Functions (BTF) they can achieve geometrical accuracy and provide a photo-realistic results. In order to obtain these results, they used 151 cameras with 12 megapixels that acquire High Dynamic Range (HDR) sequences. This configuration allows 151 simultaneous pictures with high geometrical accuracy. However, it is not possible to be used by everyone, hindering generalized use of the system.

The projects mentioned so far use acquisition hardware which reach high quality and accuracy, justified by dimension and quality required in cultural heritage projects. Other approaches, that require less accuracy in comparison with these projects, may take different acquisition hardware, such as Microsoft Kinect[®].

Software such as KinectFusion[IKH⁺11, ND11], ReconstructMe⁷ or Skanect⁸ allow controlling of one Microsoft Kinect[®] (or other low-cost cameras) through a scenario and, consequently, the acquisition of this scenario through several viewpoints. Up to now, works such as 3D Puppetry [RTHA12] used the combination of software implementations and low-cost scanners to acquire objects. This is also an example of toy-problem project, which consider controlled scenarios and use toys to create an abstraction that can be generalized to other domains.

Nevertheless, other toy-problem works have appeared recently. Miller et al. [MWC⁺12] presented a solution (Lattice-First) that also uses Microsoft Kinect[®] in order to obtain depth information of LEGO[®] blocks (Figure 1). Although this approach just acquires 3 degrees-of-freedom (DOF) and assumes that the object could not be moved out of a certain area, real-time information is guaranteed (25 frames per second). As a result, user is able to manipulate and interact with the physical object though this area. In order to achieve

these goals, pixels from user hands are segmented in acquisition phase. Color information is added afterwards, through rendering phase. This work has some shortcomings, namely the limitation of using orthogonal DUPLO[®] blocks due to Microsoft Kinect[®] low resolution and low dimensions of used blocks. Moreover, regarding the limitation of 3DOF, latitude movements are not allowed.

Other relevant work in this field is DuploTrack [GFCC12] which proposes an approach that uses instructions, similar to LEGO[®] Digital Designer⁹ virtual tool. This system also presents information in real-time and is divided in two modes: Authoring and Guidance. Authoring mode allows user to construct a physical object which digital model is being constructed and presented on the screen. On the other hand, Guidance mode instruct user in order to construct an existing object, by giving him several instructions. The representation used is equal to Miller's work, a voxel grid, but this system is able to acquire 6-DOF. User hands, depth and color information are used to segment foreground from background and to apply color to digital model. Although DuploTrack solve problems that the previous system cannot deal with, it still requires orthogonal DUPLO[®] blocks. The main reason referred is also Microsoft Kinect[®] acquisition limitations, such as noise through point clouds and low accuracy. They explain that this system also requires minimum of five blocks in order not to lose track. Otherwise, blocks can be confounded as outliers.

Our proposal requires a controlled environment and aims to identify orthogonal and non-orthogonal 3D shapes, allowing the construction of simple and complex objects. We also use a low-cost camera to acquire physical objects, with the purpose of disseminate our solution. We choose DUPLO[®] blocks because they are easy to use, educational and accessible to everybody. Moreover, we can easily access to LEGO[®] database (LDraw¹⁰) in order to create a repository.

3 SYSTEM OVERVIEW

Our proposal aimed to construct 3D objects almost in real-time, with an acceptable success rate considering the number of components known. Our system is composed by four phases: (1) acquisition of a 3D polygonal mesh; (2) segmentation of acquired object; (3) retrieval of segmented object components (sub-parts); and (4) construction of a digital model, using retrieved LEGO[®] blocks. As a result, we considered both efficient segmentation and retrieval algorithms. In order to construct the model, we used descriptors to retrieve every physical component presented in the object.

⁷ <http://reconstructme.net/>

⁸ <http://skanect.manctl.com/>

⁹ <http://ldd.lego.com/>

¹⁰ <http://www.ldraw.org/>

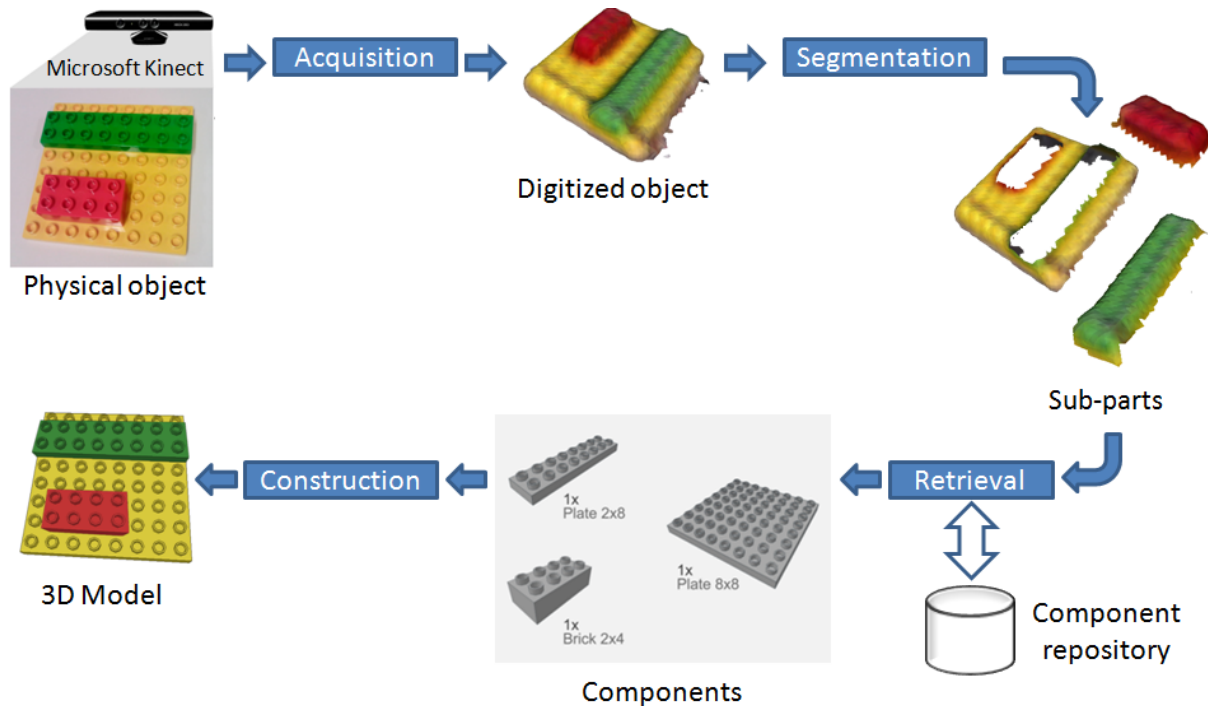


Figure 2: System overview, considering acquisition, segmentation, retrieval and construction phases.

Figure 2 describes an overview of our system, providing a construction algorithm capable to identify blocks that belong to an object. In particular, our approach aimed at going further than Lattice-First or DuploTrack, enabling the construction of blocks that are not considered in this projects (for example, curved blocks). Therefore, we pretended also a significant increase in the number of blocks that exist in the database (component repository).

Our system starts by acquiring a polygonal mesh, using Skanect. This is accomplished by moving one Microsoft Kinect® around the physical object. Using both depth and color information, we segment our object and get several sub-parts. As a result, each physical component is detached and we are ready to identify it. Subsequently, each sub-part is compared with components that exist in our repository and retrieval is performed. This process identifies physical components and makes the construction of LEGO® blocks possible.

3.1 Acquisition

We used a low-cost scanner, Microsoft Kinect®, in order to acquire physical objects (composed by DUPLO® components). The justification behind our choice, is the fact that low-cost scanners are getting used by lots of enthusiasts, and it is a recent technology that is being more and more present at people homes.

Our application used Skanect software, that provides depth and color information. The main reason behind the use of this software is because it registers all views



Figure 3: System setup. The object is created and subsequently positioned at the center of the table. Afterwards, it is acquired using one Microsoft Kinect® and able to be processed.

acquired through Microsoft Kinect®, allowing us to define a bounding box that excludes some outliers (such as walls or floor). Moreover, Skanect has the advantage of acquire color (and depth) information easily, in comparison with other approaches. However, it has some limitations: (1) it is only able to export a complete mesh (boundary representation); and (2) the resulting mesh has a huge amount of outliers, regarding the use of other objects which help Skanect not to lose track. To overcome these limitations, we used Point Cloud Library (PCL) [RC11] which is able to import the resulting mesh and convert it into a point cloud (it

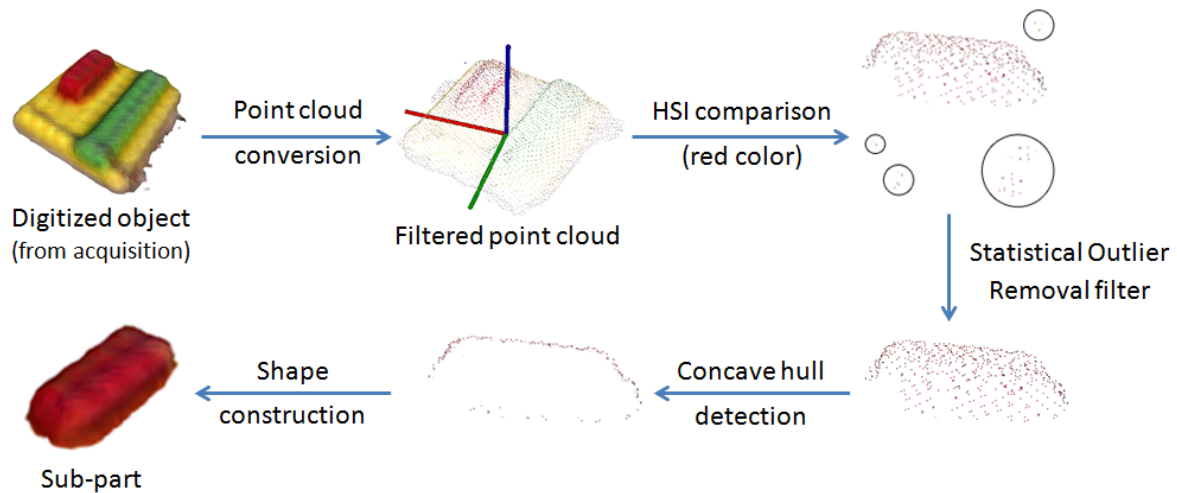


Figure 4: Example of segmentation for red color. The filtered point cloud from acquisition is used to identify the red color, through HSI comparison. We then filter outliers and detect concave hull. In the end, a shape construction is performed for every different color.

also speedup our filtering and segmentation processes). For this reason we can remove outliers and store depth and color information about our physical object.

Our setup is presented in Figure 3 where you can see a Microsoft Kinect®, several DUPLO® blocks and a turntable. Although a promising approach is to acquire objects though several table rotations, more valuable results were obtained by moving the scanner around a bounding box. Moreover, Skanect also helped us to configure our setup, providing real-time collaboration. We used an aluminium foil and transparent boxes in order to remove the maximum possible outliers and, consequently, accelerate the acquisition process.

The filtering is a process that is used by several algorithms to remove outliers from noise measures, lack of calibration, and imprecision problems due to registration. In our approach, we filter the acquired polygonal mesh in order to remove the objects added to help Skanect in registration. This process is performed by converting the acquired polygonal mesh into a point cloud. Using Principal Component analysis (PCA) [MN95], our point cloud is represented by a covariance matrix. As a result, we made a projection of the point cloud, ensuring that it is aligned with axes. The tabletop plane helped to perform this projection correctly. Therefore, we removed outliers that are behind length and width boundaries, an input of our algorithm which depend on the size of the acquired objects. Finally, we adjust the height boundary that depends on the height of the physical object, that is also an input of our algorithm. With this methodology, we reduce our point cloud from ~12 Mb to ~600 Kb, increasing the speed of our algorithm. Note that having the object aligned with the axes is determinant

to provide a successful construction, because position and orientation of virtual components depend on this.

3.2 Segmentation

The acquired color and depth information from Skanect led us to create a boundary representation and, subsequently, a point cloud. However, none of this representations allow the identification of each component that belongs to the acquired object. In other words, it is not possible to recognize what components form the point cloud.

Therefore, our segmentation phase received the filtered point cloud and aimed to divide it in several sub-parts. We used color to segment each object component, assuming that adjacent components have different colors (Figure 4).

We used algorithms available on PCL that helped to filter each component by color. Hue, Saturation and Intensity (HSI) comparison [RC11] allows the creation of a set of filters that recognize each color. It also enables the creation of a filter that removes noise produced by registration failures, by removing points that have less than a certain number of neighbors.

The output of this phase is one independent shape for each sub-part, providing important information to retrieval phase. These shapes are constructed through three steps: (1) outlier removal, through a Statistical Outlier Removal filter [RMB⁺08]; (2) concave hull creation [RC11]; and (3) shape construction [RC11]. The first step uses the segmented sub-part and removes outliers, i. e. points that have the approximately the same color, but are distant neighbors. We performed Statistical Outlier Removal filter that uses statistical analysis techniques to remove these noisy measurements. This

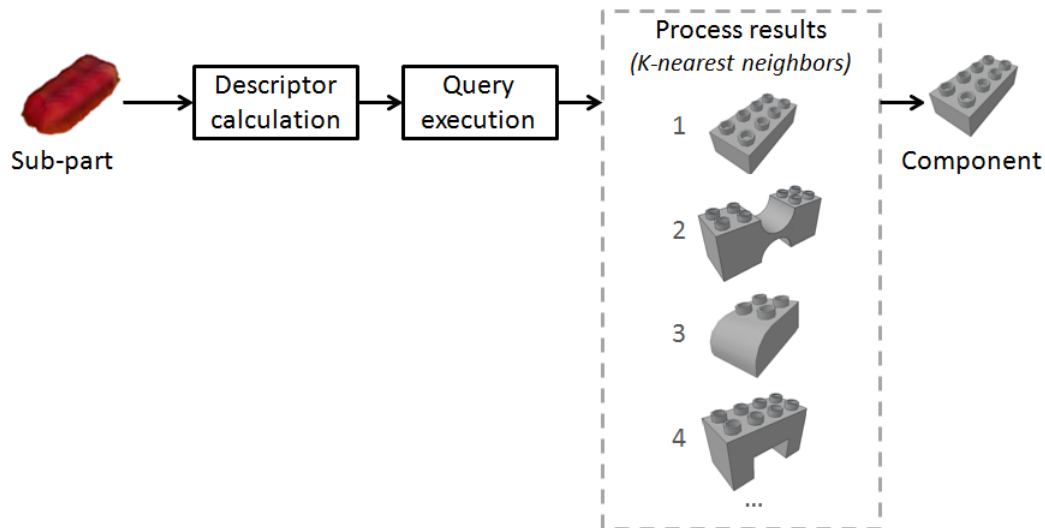


Figure 5: Retrieval phase. Our algorithm performs one query for each sub-part and returns the most similar parts.

filter is useful because acquisition outputs may have some points that are outliers due to tracking errors. We then create a concave hull in order to remove points that are not relevant for our solution. Take into account that concave hull creation tries to fill empty space created by non well acquired sub-parts, for example, due to occlusions. Finally, the shape is constructed, using the performed concave hull.

Adding to this output, there is also other information that needs to be stored for every sub-part: the color, the orientation and the centroid (one position in space). All this information is collected through filtered point cloud and required for construction phase.

3.3 Retrieval

When segmentation is concluded, we perform retrieval for each sub-part, with the purpose of constructing our model (Figure 5). Note that for every acquired physical component must exist a correspondent virtual component in our repository. First of all, one query is initiated for each sub-part. Afterwards, the descriptor is calculated for each sub-part, building one vector per part that represents that sub-part information in a more efficient way. Subsequently, the query is executed in our database, comparing the vector created with other descriptors. These descriptors were created through indexing phase and the process is similar: a descriptor was created for every virtual component that exists in our repository. The returned results of our algorithm are presented in two ways: either by best match selection or via k-nearest neighbors. The first best match selection for every sub-part is used for construction phase. On the other hand, k-nearest neighbors are used to disambiguate results, i.e, if the algorithm are not sure about one block, it can ask user using information on this list. There are several algorithms which perform index and retrieval of 3D models. Our work required a

time-efficient algorithm in order to fulfill our requirements. As a result, we chose Spherical Harmonics (SHA) [FMK⁺03] shape descriptor, that decomposes a 3D model in a collection of functions defined by concentric spheres. SHA is computed for every virtual component that exists in our repository, during offline indexing, and also for every sub-part that belongs to physical object, when retrieving.

We used a NB-Tree [FJ03], which is a powerful multi-dimensional structure, to index 70 models. This approach is efficient for high-dimensional data points, mapping those points to a 1D line through Euclidean norm. NB-Tree was relevant because it accelerates the indexing and retrieval times of our algorithm.

3.4 Construction

The sequential phases considered so far are used to construct our 3D model. In other words, we acquired a 3D physical object and segmented this components considering depth and color. Afterwards, we identified every sub-part through retrieval phase and construction is performed using this retrieved virtual components (Figure 6).

Construction is accomplished using best match selection for every block identified, regarding the information from retrieval. A 3D digital model is created taking into account centroid, orientation and color for every block (this information was given at segmentation phase). As a result, the relationships between physical components are kept. However, these relationships have some errors due to noise, taking into account that we do not deal with collisions.

3.5 Visualization and exploration

We aim to provide visual feedback of our constructed 3D model. Considering the domain of the toy-problem,

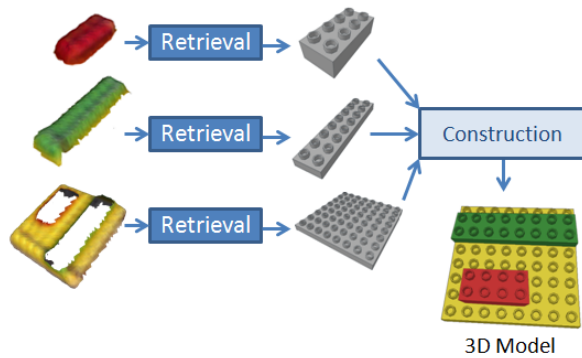


Figure 6: Construction phase.

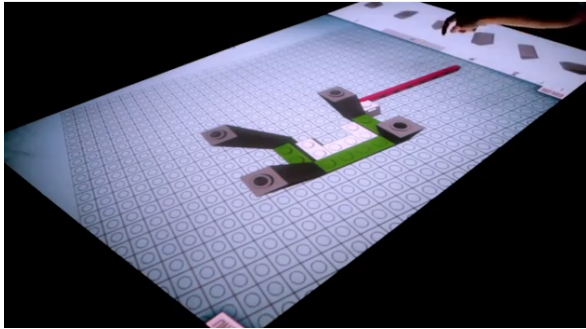


Figure 7: LTouchIT prototype. Through this application visualization and exploration of constructed models is possible.

LTouchIT [MF11] (Figure 7) application is going to be used. This application allows the construction of a 3D digital model using LEGO® blocks, through a multi-touch table. The user interacts with the table and is able to create several models, using blocks that exist in the database.

LTouchIT will represent the output of our construction algorithm, providing visual feedback and user input. It will be used mainly for user testing, allowing visualization and exploration of constructed models through several views. As a result, it helps to analyze the quality of the performed construction. However, LTouchIT application has some shortcomings regarding our approach. First, it uses a grid with lower dimensions due to the use of LEGO® blocks in spite of DUPLO® blocks. Second, the database used is incorrect, for the reasons mentioned above. Thus, some adaptation is needed in order to fulfill the requirements of our solution.

4 RESULTS

This section corresponds to the evaluation phase and obtained results of our algorithm. We divided this section in different subsections in order to explain what is evaluated and how this evaluation was performed.

4.1 Acquired models

As said before, we considered a toy-problem in order to validate our algorithm. As a result, for this partic-

ular problem we classified DUPLO® blocks through four different categories: standard blocks; additional blocks; curved blocks; and complex blocks. The standard blocks are orthogonal blocks that are considered in state-of-art projects ([MWC⁺12, GFCC12]). The additional blocks are orthogonal blocks that are not considered in standard blocks category. The curved blocks are blocks that have at least one curve but are relatively simple. Finally, the complex blocks are blocks that are not considered in other categories.

The dimensions of the acquired blocks varied between about 1.5 to 14.5 centimeters with respect to height, and about 3 to 27 centimeters with respect to the length and width. We took into consideration that we mainly have small blocks to acquire. Moreover, Microsoft Kinect® accuracy should also be considered in order to produce relevant acquisition results.

In relation to our component repository, some of the indexed models came from LDraw library, which is actualized by LEGO® enthusiasts. Because of this, we needed to be careful regarding the use of those blocks. As a result, we performed a correction of errors (such as normals) in the used models. Nevertheless, in order to have the 70 models, we also created some virtual components according to LDraw format standards.

4.2 Evaluation methodology

Our approach was evaluated considering objective appreciations. Objective measures aim to evaluate the algorithm through precision and time. Precision is the percentage of retrieved components that are relevant, i.e, number of correct components regarding the total number of components. Time is the sum of the time of all phases and intend to evaluate algorithm efficiency. Objective metrics gave us percentages and efficiency measures to analyze our algorithm.

Using the categories mentioned in last subsection, ten physical constructions were performed (Figure 8), where DUPLO® blocks vary between two to five blocks. There are two aspects that we took into account: (1) the color of each block, ensuring that two adjacent blocks could not have the same color; (2) regarding Kinect® shortcomings, we excluded transparent and bright blocks.

4.3 Discussion

For a faithful construction, it is necessary to have retrieved virtual components that match the acquired components of the physical object. However, the results achieved do not precisely construct the physical object.

The objective evaluation was performed through our ten physical constructions, and the results are summarized in Figure 9. We conclude that 22% of the time our algorithm gave the correct answer at the first time (best

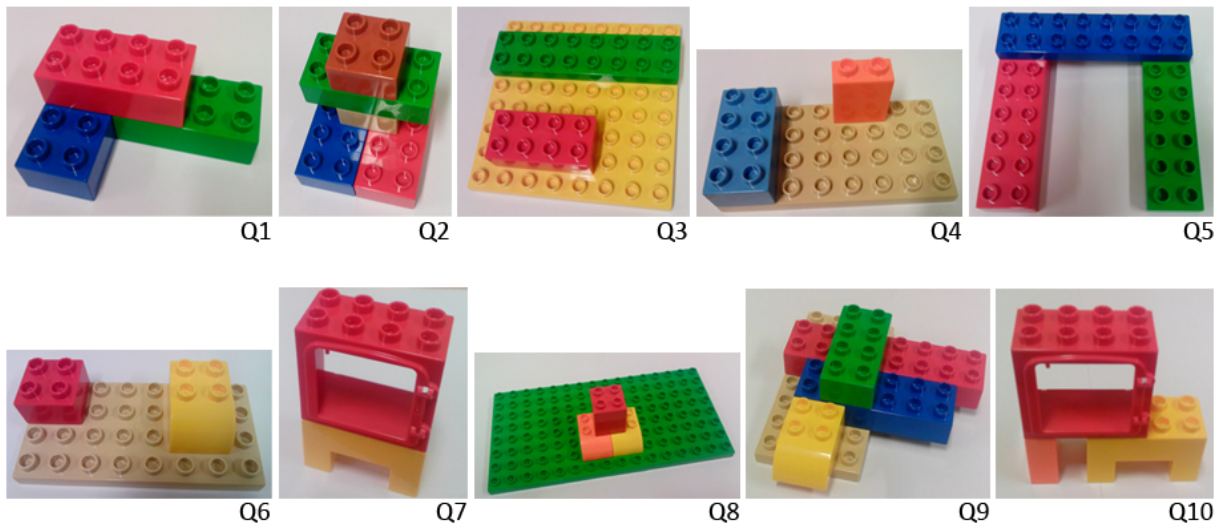


Figure 8: Acquired objects with multiple components (ten physical constructions were performed). Objects are composed by two to five blocks.

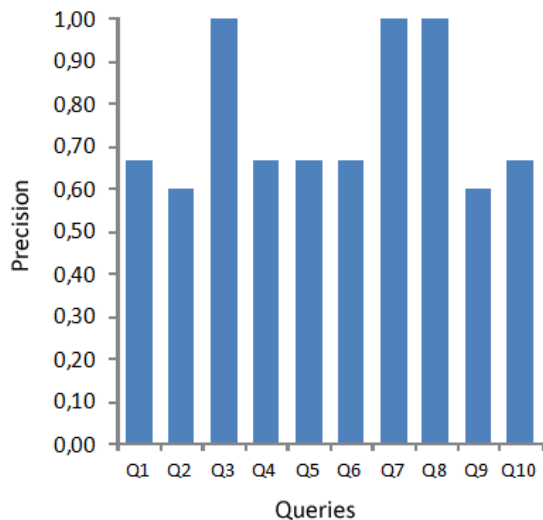


Figure 9: Objective tests: precision results. The plot represents precision obtained considering correct retrieved components within the first five results.

match selection). However, if we consider the first five results from retrieval, the correct answer was 75% of the time. This lead us to propose an application that asks if the construction is well performed and suggests the first five results if the construction is not correct at first glance. This result need to be considered because we also observed that most often plates were confused with bricks, which have slightly higher height.

In relation to time measures, we measured the total indexing time and the average time to construct an object part. Although the indexing time is done offline, the time spent to index all models in our database is quite insignificant (less than two minutes). This result

is possible due to the use of SHA descriptor. The average time to construct a 3D object is defined by the sum of all phases. We realized that average time to retrieve sub-parts may become difficult to handle only when we have several sub-parts in the acquired scene. However, with the queries performed, it does not compromise our requirements (about 7 seconds to retrieve three sub-parts). In relation to average time to construct a 3D digital model, we evaluate this time by time spent to acquire the scene and compute the mesh (thereabout 25 seconds) plus ~ 16 seconds (average) to perform the algorithm.

We conclude that the results achieved depend on the database used. First of all, the number of models in the component repository (our repository considers 70 models). Approaches referred in related work have much less models (1~3), which clearly help to retrieve object sub-parts in a more effective way. Second, there are a great amount of components in our repository that are similar among themselves. As a result, the identification of what virtual component correspond to an acquired physical component is a non-trivial challenge. In addition to this problem, the use of blocks that fit one another increase the difficulty because blocks that have other blocks above that block are not well acquired. Although we surpassed this problem by creating a concave hull for every brick that fills the empty space, it is not totally reliable.

We also confirm that it is possible to have an application that represents models almost in real-time with this approach. However, the correctness of the construction performed depends strongly on the success of the acquisition, on the accuracy and also on the repository used.

4.4 Limitations

For our problem setting, there are several limitations to overcome: (1) scanner resolution; (2) block size; and (3) Skanect acquisition results.

Low-cost scanners have some shortcomings, namely poor resolution, poor accuracy and also produce more noise. However, we hope that new advancements in this field are going to generate new technologies that decrease the problems mentioned.

The major limitation of our entire project is that the use of small blocks, allied with low resolution scanning, produces weak points clouds. Moreover, some DUPLO® blocks can cover one another preventing them from being totally captured. As a result, the generated shapes are partially incorrect and retrieved results are inaccurate.

The use of a software, such as Skanect or ReconstructMe, allows the registration of several views produced by scanner. Although the algorithms behind this kind of application are tested and are extremely efficient, they have some disadvantages, mainly for small size objects. Thus, we need to add several objects to our scene in order not to lose track. Afterwards, we got greater files to process and filtering are not trivial.

5 FUTURE WORK

Microsoft® is now working on Kinect 2.0, which have higher resolution when compared with the current version. Moreover, the RGB camera is upgraded from 34-bit RGB to 16-bit YUV, and a new infrared sensor is added. Therefore, we hope that using this configuration is going to help to acquire objects more efficiently.

In relation to segmentation, our algorithm needs an upgrade that allows to acquire at least two blocks with the same color. In order to achieve this, we will need to use a segmentation algorithm such as Region Growing. An upgrade that segments more different colors can also be added.

One part of our project consists on the integration of LTouchIT, providing an application that allows manipulation of acquired 3D objects. This is going to help users to evaluate our algorithm through a Likert scale. This evaluation is required because, although some results are not precisely correct, they may be close to what would be expected. Moreover, this will lead to ask if the presented result is correct and, if it is not correct the application will suggest the first five results (from retrieval phase). This process can be an automated task which is possible due to fine-tuning of our algorithm in combination with other methodologies.

We also consider to let users evaluate our algorithm, through subjective measures. On one hand, objective metrics gave us percentages and efficiency measures. On other hand, we want to know how acceptable could

be one result in terms of quality. This evaluation is relevant because, although some results are not precisely correct, they may be close to what would be expected.

Our solution uses a toy-problem, LEGO® blocks, to demonstrate the construction algorithm. We would like to propose several proof-of-concept applications that include a similar approach, namely for health, mechanical or electric industries. For example, in mechanical industry, the identification of several physical components that belong to the car, such as radiator and engine. For all mentioned applications, the concept of learning can be improved to 3D and many appliances can be done. Imagine if I could scan a set of bones of a leg and add them virtually to other structure, using a construction algorithm to provide help.

6 CONCLUSION

This research is focused on identification of object components in order to construct a digitized 3D model. Our approach considers newly low-cost technologies, such as Microsoft Kinect® and the increase in computing power that allows storage and faster data availability. As a result, some efficient retrieval algorithms have appeared. Having in mind the recent technological boost, our solution aims to identify all physical components that belong to an object. The main contribution of this work is an application that performs a segmented construction of a 3D physical object acquired through a low-cost device, using color and shape information.

In comparison with similar approaches, our solution has the advantage of having a large number of components allowed to be identifiable, from a repository. In particular, our repository guarantees that 70 different physical components can be acquired through a low-cost device and detectable with our algorithm. However, based on the fact that blocks are covering each other, the precision achieved is reduced.

The experiments clearly indicate that small size blocks are complex to analyze through Microsoft Kinect® acquisition, taking into account its low accuracy. Subsequently, the produced point cloud is insufficient to identify physical components correctly. Therefore, the obtained results are not yet enough to generalize our approach. However, we consider that our solution may be applicable to many other domains if those physical components are larger in comparison with the size of our acquired blocks. Moreover, if the components in the repository were more different among themselves, it would produce more accurate results.

7 ACKNOWLEDGMENTS

This work was partially supported by FCT through the PIDDAC Program funds (INESC-ID multiannual funding), reference PEst-OE/EEI/LA0021/2013, and through the project 3DORuS, reference PTDC/EIA-EIA/102930/2008.

8 REFERENCES

- [AB94] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994.
- [AFS06] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 2006.
- [BR02] Fausto Bernardini and Holly Rushmeier. The 3D Model Acquisition Pipeline. *Comput. Graph. Forum*, 21(2):149–172, 2002.
- [CTSO03] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3D model retrieval. *Computer Graphics Forum*, 22(3):223–232, 2003.
- [FJ03] MJ Fonseca and JA Jorge. NB-Tree: An indexing structure for content-based retrieval in large databases. *Technical report*, pages 1–25, 2003.
- [FMK⁺03] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A search engine for 3D models. *ACM Transactions on Graphics*, 22(1):83–105, 2003.
- [GFCC12] Ankit Gupta, Dieter Fox, Brian Curless, and Michael Cohen. DuploTrack: a real-time system for authoring and guiding duplo block assembly. In *Proceedings of 25th ACM Symposium on User Interface Software and Technology (UIST)*, pages 389–401, 2012.
- [IKH⁺11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard A. Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew J. Davison, and Andrew W. Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In Jeffrey S. Pierce, Maneesh Agrawala, and Scott R. Klemmer, editors, *UIST*, pages 559–568. ACM, 2011.
- [LRG⁺00] Marc Levoy, Szymon Rusinkiewicz, Matt Ginzton, Jeremy Ginsberg, Kari Pulli, David Koller, Sean Anderson, Jonathan Shade, Brian Curless, Lucas Pereira, James Davis, and Duane Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. *SIGGRAPH*, pages 131–144, 2000.
- [LSFG11] H Laga, T Schreck, A Ferreira, and A Godil. Bag of words and local spectral descriptor for 3d partial shape retrieval. *Proceedings of the Eurographics Workshop on 3D Object Retrieval (3DOR'11)*, pages 41–48, 2011.
- [MF11] Daniel Mendes and Alfredo Ferreira. Virtual LEGO Modelling on Multi-Touch Surfaces. *Visualization and Computer Vision (WSCG)*, 2011.
- [MMWG11] Markus Mathias, Andelo Martinovic, Julien Weissenberg, and Luc Van Gool. Procedural 3D Building Reconstruction Using Shape Grammars and Detectors. *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 304–311, 2011.
- [MN95] Hiroshi Murase and Shree K. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14(1):5–24, 1995.
- [MWC⁺12] Andrew Miller, Brandyn White, Emiko Charbonneau, Zach Kanzler, and Joseph J LaViola. Interactive 3D model acquisition and tracking of building block structures. *IEEE transactions on visualization and computer graphics*, 18(4):651–9, 2012.
- [ND11] RA Newcombe and AJ Davison. KinectFusion: Real-time dense surface mapping and tracking. (*ISMAR*), pages 127–136, 2011.
- [RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.
- [RMB⁺08] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 2008.
- [RTHA12] Brian Curless Robert T. Held, Ankit Gupta and Maneesh Agrawala. 3d puppetry: A kinect-based interface for 3d animation. *Proceedings of the 25th annual ACM symposium adjunct on User interface software and technology*, 2012.
- [STK02] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of Polyhedral Surfaces using Decomposition. *Computer Graphics Forum*, 21(3):219–228, 2002.
- [SW11] C Schwartz and M Weinmann. Integrated high-quality acquisition of geometry and appearance for cultural heritage. In *proceedings of The 12th International Symposium on Virtual Reality, Archeology and Cultural Heritage*, 2011.
- [TDVC11] Hedi Tabia, Mohamed Daoudi, Jean-Philippe Vandeborre, and Olivier Colot. Deformable shape retrieval using bag-of-feature techniques. *Proceedings of the 3D Image Processing (3DIP'11)*, 2011.
- [VMC96] T Varady, RR Martin, and Jordan Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, pages 1–28, 1996.

A framework for detection of linear gradient filled regions and their reconstruction for vector graphics

Ruchin Kansal
Department of computer
science
IIT Delhi
India, New Delhi
rkansal@adobe.com

Prof. Subodh Kumar
Department of computer
science
IIT Delhi
India, New Delhi
subodh@cse.iitd.ac.in

ABSTRACT

Linear gradients are commonly applied in non-photographic art-work for shading and other artistic effects. It is sometimes necessary to generate a vector graphics form of raster images comprising such art-work with the expectation to obtain a simple output that is further editable, say, using any popular vector editing software. Further, this vectorization process should be automatic with minimal user intervention. We present a simple image vectorization algorithm that meets these requirements by detecting linear gradient filled regions and reconstructing the vector definition using that information. It uses a novel interval gradient optimization scheme to derive large regions of uniform gradient. We also demonstrate the technique on noisy images.

Keywords

Image vectorization Gradient reconstruction Region detection

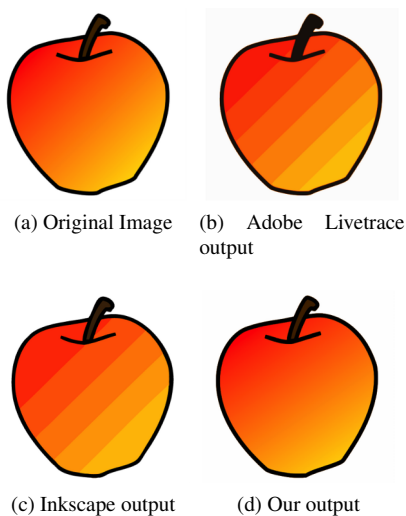


Figure 1: Comparison of different outputs.

1 INTRODUCTION

The advent of new mobile and touch devices has motivated designers to create and publish their content in vector format. Vector graphics represents 2D images in terms of mathematical elements like curves, contours, straight lines and other shapes, along with their attributes such as fill, stroke, transparency and so on. This is different from raster representation, which stores a color sample at each pixel center. Vector graphics supports rasterization on the fly and therefore it can be viewed at different scales and resolutions without any artifacts.

With the increasing demand for vector content, the need for converting raster images into vectors has also increased. This process is called *Vectorization*. We set out to obtain automatic vectorization with minimal human intervention even on potentially noisy images. Later re-rasterization of our vector representation should produce the appearance of the original image at various scales. Further, for wide-spread usage, this vector representation should be in a standard form and be editable. However, it is not easy for an algorithm to meet all these conditions for every image. In fact, these may be conflicting goals. For example, to match the vector appearance with the original image, an algorithm might generate smaller patches due to which editing becomes difficult. Further, application of color gradients for shading effects in a non-photographic image poses additional challenges to vectorization. In this case, the vectorization algorithm must derive the gradient definition and use it to approximate the color information. Many algorithms, including commercial software [27, 1, 13], are unable to appropriately reconstruct such gradients. For example, they may approximate the gradients with patches of constant fill regions. (See Inkscape [13] and Adobe Illustrator [1] examples in figures 1c and 1b respectively).

A few others, including the one titled ARDECO [17], focus on computing the gradient. For example, ARDECO uses first and higher order gradients to store the color information. However, these techniques either generate too many vector patches or their

representations are so complex that their output cannot be easily edited.

Vector graphics can be represented using an open vector format such as *EPS*, *PDF*, or *SVG*[28] or it could be a proprietary format (such as *Adobe Illustrator* or *Corel*). Among open formats, *SVG* is possibly the most widely used vector format for web and digital media, which we have chosen as our output. Nonetheless, the definition of linear gradient is largely the same in all vector standards modulo occasional minor differences. *SVG* specifies *linear color gradient* as continuous smooth color transition along a 2D direction from one given color at a known position to another. This direction is called the *Gradient Vector*. The value of each pixel along the gradient vector may be calculated by linearly interpolating the two end colors. The *gradient normal* is orthogonal to the gradient vector. The color of each pixel on the gradient normal remains same. The *SVG* standard also allows fixing of more than two colors along the gradient vector, to form a smooth multi-color transition. These specific points on the gradient vector with pre-defined color values are called *Gradient Stops* (See figure 2).

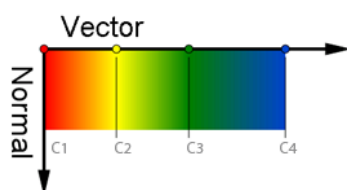


Figure 2: Linear Gradient defined by four gradient stops(C1, C2, C3 and C4). Notice the color along the gradient vector is defined by linear interpolation from one stop to another but the color of the pixels along the gradient normal remains same.

We have developed an approach that can detect linear gradient filled regions as well as the gradient values. While the contributions of this paper are primarily in effective recovery of regions with uniform gradient, for completeness we do also produce boundary curves and regions with uniform fill color where necessary. We do not target vectorization of photographic quality images, but rather art-design by artists. The distinguishing feature of such images is that they contain relatively large areas of uniform fill and gradients but suffer from noise and other smoothing and post-processing artifacts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

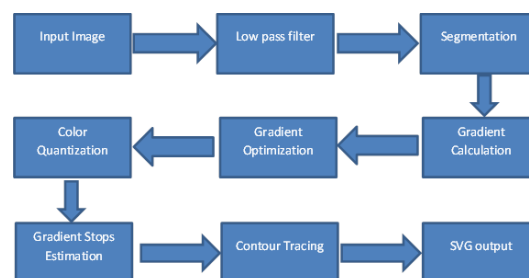


Figure 3: The complete pipeline

2 PREVIOUS WORK

Vectorization generally involves some form of image segmentation followed by a vector approximation for each segment. Significant research in both vectorization and, more generally, image segmentation exists. The research in vectorization is done with various objectives such as fitting smooth curves [15, 24, 2, 9], minimizing the number of colors used [29], preserving editability [4] and matching appearance with input [17, 25, 16, 21]. Image segmentation prior to vectorization is commonly based on edge detection [6], color quantization [8, 11, 5] or a global optimization function [19, 18, 20] to reduce the overall energy.

Early work in this field focused on line drawings and bi-tonal images [6, 9, 10]. These approaches are mainly based on edge detection [6], thresholding [7, 14, 22, 23], thinning [26] and contour tracing [12]. The extracted line, image contour or region is represented by vector graphics primitives, e.g., curves and paths. More recent algorithms [17, 25, 16, 21, 29, 4, 15] deal with full color images and their goal is to generate high fidelity output.

ARDECO[17], proposed by Lecot and Levy in 2006, tries to decompose the input image into patches. Each patch is approximated by a constant color, linear or higher order gradient in order to minimize the overall energy. The energy function in their approach is determined by a boundary length function and a curve fitting term. The weighting of terms is controlled by user input. Since the energy functional is quite generic, it can handle images with fine details. At the same time it often produces a large number of patches and consequently it is not possible to edit the final vector graphics easily for post-editing. Further, for large gradient fill regions, it often fails to converge to any result. Also, due to linear constraints the segment boundaries produced by them is often not smooth. Finally, the user needs to adjust several parameters by experimentation. Our algorithm is simpler than *ARDECO* as it considers only first order gradients. Further, our algorithm produces fewer regions so that a user can edit the image easily.

Sun et al [25] introduced a vectorization approach using *Gradient Meshes*. There, a gradient mesh is defined by a grid using topologically planar rectangular

Ferguson patches with mesh-lines. Control points of the mesh have three attributes: position, derivative and color. Their approach relies on user assistance for mesh initialization and placement. Recently, Lai, Hu and Martin, 2009 [16] improved that algorithm by generating the gradient mesh automatically. The output of gradient mesh is quite impressive and it can even be applied to photographic images. However, the size of their representation is too unwieldy for further editing. Moreover, Gradient Mesh is not a standard primitive and are hence less portable. They cannot be rendered or edited by standard tools.

Price and Barrett [4] proposed an approach for interactive image editing using object based vectorization. They allow the user to select an object and then graph cut is used recursively to form a hierarchical object tree. For each object they define a mesh by locating the corner points and doing recursive subdivision. The resulting mesh can be edited by various tools. However, the approach is designed to be driven by user manually. Also, the algorithm does not handle gradient reconstruction, it only provides a better means of object construction.

Diffusion curves[21] is a different approach to represent smooth shaded images. A diffusion curve partitions the space through which it is drawn, defining different colors on either side. These colors may vary smoothly along the curve. In addition, the sharpness of the color transition from one side of the curve to the other can be controlled. Due to the limitations with Poisson equations, the color variations in all raster images may not be represented by this system, especially when the image has sparse features in some areas.

Xia et al [29] proposed a vector based representation in which the image is decomposed into non-overlapping triangular patches with curved boundaries. The color variation over each triangular patch is approximated with a thin-plate spline for every color channel. It allows them to approximate raster images with both smooth variations and curvilinear features. Although, the representation is powerful and compact, again editability and portability is a concern.

A nice discussion of vector primitives related to color gradients is provided by Pascal et al. [3]. They describe various available techniques for construction and rendering of such vector primitives. They mention the current methods of vector creation by hand as well as through vectorization. Some practical challenges and limitations of these approaches are also explained.

Adobe live trace [1] and Vector magic [27] are popular commercial applications, which are used for vectorization and produces standard vector graphics. Inkscape [13] is an open source tool which is also used for vectorization and vector editing. However, none of these packages recognizes linear color gradients in the image

and therefore such regions are approximated by rectangular stripes (See figure 1).

3 OUR APPROACH

Figure 3 depicts our pipeline. We start with a raster image. In our experiments all input images are 8-bit per-channel RGB images but the technique is independent of the color format. Like most vectorization approaches, we first segment a filtered version of the image. Color discontinuity imposes segment boundaries. Next, for each segmented region we determine if it can be represented by a linear gradient function. This decision is made by searching for a gradient value that is supported by all pixels of the region. In particular, we calculate the range of gradient values supported by each pixel. A global optimization across pixels of the region then determines the most plausible gradient for the region. Finally, using this optimized gradient direction, we find the gradient stops within the region. In terms of figure 3, the main contributions of our algorithm are in stages 4, 5 and 7.

For regions that cannot be represented using a linear gradient, we employ a color quantization approach to minimize the number of vector elements. Each region is then vectorized using the potrace engine [24] (although any vectorization approach would suffice). Potrace is designed to generate smooth contours of the features which works well with our pipeline. Each stage of the pipeline is described next.

3.1 Image Smoothing

To reduce the effect of noise in the image, we apply a Gaussian blur of radius 3. It reduces the sharpness near edges and produces a relatively smooth image.

3.2 Initial Segmentation

As a preprocessing step, we first perform image segmentation using a standard scheme. We have used mean shift segmentation followed by a flood fill, which gives us good result. although a more specialized segmentation algorithm can also be employed.

3.3 Gradient Approximation

In order to determine the gradient direction m we reconstruct values from the given samples. However, we must also consider that the input color values are imprecise due to noise, smoothing and rasterization. Due to imprecise color values at pixel centers, we resort to an interval color scheme. In particular, if the color at pixel p input to this stage is c , we allow that the actual color is in the range $[c_- : c_+]$, where $c \in [c_- : c_+]$. For example, if c only has rounding error, $c_- = c - 0.5$ and $c_+ = c + 0.5$.

If the gradient slope at pixel p is m , we expect the color at the gradient normal $p + k \frac{1}{m}$ within a segment to be in

Algorithm 1 Find the slope range

```

1: procedure FINDRANGE( $p, R$ ) ▷ Calculate range of
   slopes for pixel  $p$  over region  $R$ 
2:    $p.range \leftarrow \emptyset$       ▷ Initialize the range of  $p$  as
   empty
3:    $S \leftarrow$  Boundary pixels of  $R$  ▷ Initialize vector  $S$ 
   with boundary pixels of  $R$  in such an order that all
   consecutive pixels in  $S$  are neighbors in  $R$ 
4:    $i \leftarrow 0$ 
5:    $c \leftarrow p.color$ 
6:   for  $i < S.size - 1$  do
7:      $p_1 \leftarrow S[i]$ 
8:      $p_2 \leftarrow S[i + 1]$   ▷ Get two neighbor pixels
   from  $S$  in  $p_1$  and  $p_2$ 
9:     if  $[c_- : c_+] \cap [p_1.color, p_2.color] \neq \emptyset$  AND
    $p.region = p_1.region = p_2.region$  then
10:       $L_1 \leftarrow line(p_1, p)$  ▷ Find line  $L_1$  passing
   through  $p_1$  and  $p$ 
11:       $L_2 \leftarrow line(p_2, p)$ 
12:       $p'_1 \leftarrow intersection(L_1, R)$       ▷ Find
   intersection of  $L_1$  with  $R$ 
13:       $p'_2 \leftarrow intersection(L_2, R)$ 
14:      if  $[c_- : c_+] \subseteq [p'_1.color, p'_2.color]$  then
15:         $p.range \leftarrow [slope(L_1), slope(L_2)]$ 
16:        break;
17:      end if
18:    end if
19:     $i \leftarrow i + 1$ 
20:  end for
21: end procedure

```

the interval $[c_- : c_+]$ for all values of k within a range, if the input color at p is c . However, due to the rasterization in the input image we may not have samples available for any value of k .

We consider a contour around p and locate the normal line passing through p on this contour. For example, this contour can be a rectangle R . Our goal is to locate the range $[c_- : c_+]$ on R . Assuming color interpolation along R , we find two samples p_1 and p_2 on R such that the color c_1 at p_1 and color c_2 at p_2 span the range $[c_- : c_+]$, where p_1 and p_2 are the closest such pixels along the contour. In other words, $[c_- : c_+] \subseteq [c_1 : c_2]$. We conjecture that the normal line intersects the rectangle between p_1 and p_2 . As an aside, if one were to search for the exact value c reconstructed from samples near p_1 and p_2 , it would yield unreliable estimates for m that are often inconsistent with the estimates of p 's neighbors.

If p is not on the boundary of its region, a pair (p_1, p_2) implies the existence of another pair (p'_1, p'_2) on the opposite side of the rectangle. For pair (p_1, p_2) , we form straight lines by joining p_1 and p , and similarly by joining p_2 and p (Figure 4 explains this construction, see the blue and red lines passing through p). The intersec-

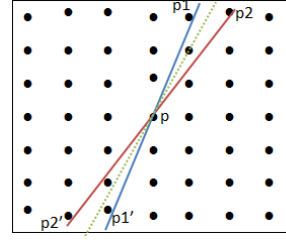


Figure 4: The setup: A rectangular grid of pixels around p is considered. The color interval of pixel p , $[c_- : c_+]$, lies in the colors at pixels p_1 and p_2 . This implies the normal direction through p , passes between p_1 and p_2 . Consider lines joining p with p_1 and p_2 respectively. These lines intersect at the opposite side of the grid on p'_1 and p'_2 . If $[c_- : c_+]$ is spanned by the colors at p'_1 and p'_2 , the normal directions is assumed to lie between the two solid lines. Additionally, the green dotted line is formed by fitting a line among all pixels whose colors are similar to that of pixel p . This estimated slope is also stored for each pixel p .

tions of these lines with the opposite boundary of rectangle R provides the conjugate pair (p'_1, p'_2) . Again, we need not have samples available at p'_1 and p'_2 , unless R is symmetric about p . We reconstruct the color, respectively, c'_1 and c'_2 at positions p'_1 and p'_2 from the neighboring samples. If again $[c_- : c_+] \subseteq [c'_1 : c'_2]$, it is evidence of the normal line passing between p'_1 and p'_2 . If the slopes of lines $p_1p'_1$ and $p_2p'_2$ are $\frac{1}{m_1}$ and $\frac{1}{m_2}$, respectively, we say that pixel p favors a color gradient in the range $[m_1 : m_2]$ subject to the condition that pairs (p_1, p_2) and (p'_1, p'_2) lie in the same image region. Please note that if the range $[p'_1 : p'_2]$ is not tight and its subset contains the color range $[c_- : c_+]$, that subset is used instead to provide a tighter gradient range. This approach is presented in algorithm 1.

Not the entire range of gradients $[m_1 : m_2]$ is equally favored by p . We also estimate the most favored gradient m' and weight a value $m \in [m_1 : m_2]$ by its difference from m' . To find m' , we compute the best fit line to the color values nearest c within R . In particular, we form a set of points S including every pixel within rectangle R with color within $[c_- : c_+]$. The calculation of favored slope is explained in algorithm 2.

If a pixel does not produce a gradient range, either it is not a part of a gradient filled region, or it cannot provide candidate gradients due to noise in the image. On the other hand, it is possible for a pixel to provide multiple gradient candidates due to noise. We include all ranges in the optimization process described next.

Every pixel p_i of a presumed gradient fill region similarly produces its favored gradient m'_i and slope range (m_{i_1}, m_{i_2}) . We choose a single gradient value for the region that best satisfies all ranges.

Algorithm 2 Find the favored gradient slope

```

1: procedure FINDGRADSLOPE( $p, R$ )  $\triangleright$  Calculate
   favored gradient slope of pixel  $p$  over region  $R$ 
2:    $p.slope \leftarrow nil$ 
3:    $Q \leftarrow$  all pixels of  $R$   $\triangleright$  Initialize vector  $Q$  with
   all pixels of  $R$ 
4:    $i \leftarrow 0$ 
5:    $S \leftarrow \emptyset$ 
6:   for  $i \neq Q.size$  do  $\triangleright$  Loop on all pixels in  $Q$ 
7:      $q \leftarrow Q[i]$ 
8:     if  $p.color \in [q.color_- : q.color_+]$  AND
        $p.region = q.region$  then
9:        $S \leftarrow S \cup \{q\}$ 
10:    end if
11:     $i \leftarrow i + 1$ 
12:  end for
13:  if  $S \neq \emptyset$  then
14:     $L \leftarrow FitStraightLine(S)$ 
15:     $p.slope \leftarrow slope(L)$ 
16:  end if
17: end procedure

```

Choice of this region R is important as it should be sufficiently large to have enough samples to reproduce the reliable gradient range and value. The size of region R may also vary on each pixel depending on the noise in the image. Therefore, we choose rectangles of dynamically varying sizes whose dimensions are decided on each pixel. We start with a size of 3×3 , and keep on increasing this region until the line fit error is below a certain threshold ϵ . Because of this dynamically sized region, our approach can handle different kinds of noisy images successfully.

3.4 Gradient Optimization

After computing the favored gradients for each pixel p_i , m'_i and the range $[m_{i1} : m_{i2}]$, the final gradient m_r for the region should ideally lie in this range and as close to m'_i as possible. We compute m_r by optimizing across all pixels of the region. This optimization can be easily posed and solved using a simple geometrical construction.

We select a function which maximizes its potential if the selected gradient $m_r = m'_i$. This potential monotonically decreases as m_r grows apart from m'_i . One can select a Gaussian or radial basis function as the weight, but a cosine-linear weight function provides the best results. Given two line slopes m' and m'' , the dot product of the vectors in their respective directions gives a projection of a vector on another. By definition, the magnitude of the dot product of two unit vectors decreases as the angle between them grows. We define our objective function to maximize the sum of these dot products. To do so, the value of objective function $f(x)$ is

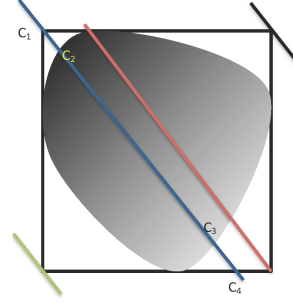


Figure 5: Gradient Stops Estimation: The shaded area is a gradient filled region while its bounding box is marked as black rectangle. We draw lines from four corners of the bounding box parallel to gradient axis (shown in different colors), since the line passing from top-left corner (marked in blue) overlaps the maximum pixels of the region, it is used for gradient stops estimation. Two stops are generated where line hits the bounding box (C1 and C4) while two stops are generated where line intersects the regions (C2 and C3). Also, note that value of C2 and C3 is determined by using the pixel color at the respective location of image while value of C1 and C4 is computed by extrapolation of C2 and C3 along the gradient axis.

computed by finding all pixels p_i which have the range $[m_{i1} : m_{i2}]$ containing x and performing a summation over the dot products with their favoured slope m'_i . i.e. $f(x) = \sum_{i=0}^n |g(x, i)|$ where $g(x, i) = \hat{x} \cdot \hat{m}'_i |m_{i1} \leq x \leq m_{i2}$ and \hat{x} and \hat{m}'_i are two unit vectors in the direction of x and m'_i respectively.

The linear optimization can be performed using any standard technique like the dynamic system based global optimization [19].

3.5 Gradient Stops Estimation

We use a heuristic approach to find the gradient stops. If we draw lines parallel to the computed gradient vector m_r from each corner of the bounding box of the region as shown in Figure 5, the one with the largest overlap with the region may be selected for stops estimation. We generate four color stops on the gradient vector, two on each end points on the bounding box and two on each intersection of this line with the region (See Figure 5). If the points at C1 and C4 do not lie in the region, their colors may be estimated using extrapolation from C2 and C3 as shown.

3.6 Color Quantization

Segmentation for the remaining colored regions of the image is performed using color quantization [11]. A color palette of the given number of colors is first prepared, and then each pixel is assigned the index of color palette that it best matches. For our experiments we

have used an octree based color quantization approach [8, 5].

3.7 Contour Tracing

Tracing is the process of fitting curves that bound each image regions. After tracing, we obtain a set of curves that represent the image geometry. We employ the potrace engine developed by Peter Schilinger [24] for this outline tracing. The same engine is also used by open source vector drawing package Inkscape [13].

3.8 Final Vector Output

Once we obtain the curves outlining each image segment, we apply fills to these curves and generate the final vector representation (in the SVG format). The region color, as noted before, may be either a solid fill color obtained through quantization or a linear gradient produced by the optimization algorithm.

4 RESULTS AND VALIDATIONS

We analyzed the vector output of our algorithm from various perspectives like appearance, editability, accuracy and error per pixel. Results are given below.

4.1 Appearance

We applied our approach to different non-photorealistic raster images and the results are presented in Figure 6.

4.2 Comparison with ARDECO

The implementation of ARDECO is publicly available on the authors webpage. In figure 9 ARDECO produced more than 1200 paths while our approach outputs 4 paths only. This is because we perform an early segmentation and then apply gradient detection on the various segments. We are also able to handle noisy images, as shown in figure 9 where the input image contains random RGB noise.

4.3 Editability

The output SVG can be easily edited using any standard vector graphics tool like Inkscape. Examples are shown in figure 7.

4.4 Accuracy

To measure the accuracy, we applied our algorithm to images whose gradient direction and magnitude was already known. The results are shown in figure 8.

<i>Input</i>	<i>Our Error</i>		<i>Inkscape Error</i>	
	<i>Gradient</i>	<i>Solid</i>	<i>Gradient</i>	<i>Solid</i>
6e	7.7	16.25	17.49	24.40
6k	11.4	13.2	18.64	21.31
6c	11.63	18.67	14.76	26.55
6i	15.64	25.14	26.20	44.8
6a	25.51	34.26	46.92	37.40

Table 1: We calculated the per pixel error for gradient and solid colored regions separately in our output and then compared with the corresponding region error in inkscape.

4.5 Per Pixel Error

To analyze the per pixel error in our output, we rasterized our vector output and then compared it with the original image to compute root mean squared error per pixel. Table 1 compares the error in our gradient and solid colored regions with the corresponding regions in Inkscape output. Both Inkscape's and our approach used a quantization palette size of 16 colors.

Table 1 shows that even the per pixel error for solid colored regions is low with our approach as compared to Inkscape. This is due to the fact that our approach excludes the gradient region while performing quantization, therefore with the same size of color palette, more accurate colors are represented.

The high per pixel error in output can be explained due to the several factors:

1. Vector and raster spaces are not equivalent. The pixel at location (x, y) in input image may not be present at the same exact location in the vector space.
2. Input image may contain small pixel level features, which are merged in larger regions during vectorization.
3. Vector output is optimized to be represented with fewer colors using some method of color minimization.

5 LIMITATIONS AND FUTURE WORK

We have proposed an algorithm to find gradient in images that optimizes the gradient values across noisy pixels. It mainly targets reconstruction of simple art drawings that can then be further edited or stylized. The proposed algorithm works well when the linear gradient in input image is specified by two color stops. Otherwise the the gradient region may be split into multiple smaller regions. This limitation can be easily handled by modifying the gradient stops estimation step to account for multiple color stops. Our approach may also not produce good results when the linear gradient is applied on small width regions, like linear gradient

on a single pixel wide curve, for it needs to find segments with a few neighbors around its pixels. We believe the algorithm can be easily adapted to handle non-linear gradients – for example a radial gradient. Our algorithm is designed to operate on each pixel independently, therefore it can parallelize well. Future work should also include deriving vector graphics for videos and using the level of optimization in a feedback loop to refine the segmentation potentially producing even fewer patches.

6 REFERENCES

- [1] Inc. Adobe Systems. Adobe illustrator cs5, 2010.
- [2] Autotrace. An open-source program for converting bitmap to vector graphics, 2004.
- [3] Pascal Barla and Adrien Bousseau. Gradient art: Creation and vectorization. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, volume 42 of *Computational Imaging and Vision*, pages 149–166. Springer London, 2013.
- [4] William A. Barrett and Alan S. Cheney. Object-based image editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 777–784, New York, NY, USA, 2002. ACM.
- [5] Dan S Bloomberg. Color quantization using octrees. 2008.
- [6] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.
- [7] Jung-Shiong Chang, Hong-Yuan Mark Liao, Maw-Kae Hor, Jun-Wei Hsieh, and Ming-Yang Chern. New automatic multi-level thresholding technique for segmentation of thermal images. *Image and Vision Computing*, 15(1):23 – 34, 1997.
- [8] D. Clark. Color quantization using octrees. *Dr. Dobbs's Journal*, pages 54–57, Jan 1996.
- [9] Dov Dori, Senior Member, and Wenyin Liu. Sparse pixel vectorization: An algorithm and its performance evaluation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21:202–215, 1999.
- [10] Kuo-Chin Fan, Den-Fong Chen, and Ming-Gang Wen. A new vectorization-based approach to the skeletonization of binary images. In *ICDAR*, pages 627–630. IEEE Computer Society, 1995.
- [11] Michael Gervautz and Werner Purgathofer. A simple method for color quantization: Octree quantization. In *New Trends in Computer Graphics*. Springer Verlag, Berlin, 1988.
- [12] O. Hori and S. Tanigawa. Raster-to-vector conversion by line fitting based on contours and skeletons. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 353 – 358, oct 1993.
- [13] Inkscape. An open source linux/windows scalable vector graphics editor, 2010.
- [14] Ralf Kohler. A segmentation system based on thresholding. *Computer Graphics and Image Processing*, 15(4):319 – 338, 1981.
- [15] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 99:1–99:8, New York, NY, USA, 2011. ACM.
- [16] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):85:1–85:8, July 2009.
- [17] Gregory Lecot and Bruno Levy. Ardeco: Automatic region detection and conversion. In *Eurographics Symposium on Rendering*, 2006.
- [18] Musa Mammadov, Alexander Rubinov, and John Yearwood. Dynamical systems described by relational elasticities with applications. *Continuous Optimization*, pages 365–385, 2005.
- [19] Musa A Mammadov. A new global optimization algorithm based on dynamical systems approach. In *Proceedings of the 6th International Conference on Optimization: Techniques and Applications (ICOTA' 04)*. Ballarat, Australia, 2004.
- [20] University of Ballarat. Ganso library for optimization functions.
- [21] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 92:1–92:8, New York, NY, USA, 2008. ACM.
- [22] Arnulfo Perez and Rafael C. Gonzalez. An iterative thresholding algorithm for image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(6):742 – 751, nov. 1987.
- [23] N. Ramesh, J.-H. Yoo, and I.K. Sethi. Thresholding based on histogram approximation. *Vision, Image and Signal Processing, IEE Proceedings -*, 142(5):271 – 279, oct 1995.
- [24] Peter Selinger. Potrace: a polygon-based tracing algorithm, 2003.
- [25] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.
- [26] H. Tamura. A comparison of line thinning al-

- gorithms from digital geometry viewpoint. In *Proceedings of the Fourth Int'l Joint Conf Pattern Recognition*. Kyoto, Japan, 1978.
- [27] Inc. Vector Magic. Vector magic, 2010.
- [28] SVG working group. Svg format for vector graphics.
- [29] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 115:1–115:10, New York, NY, USA, 2009. ACM.

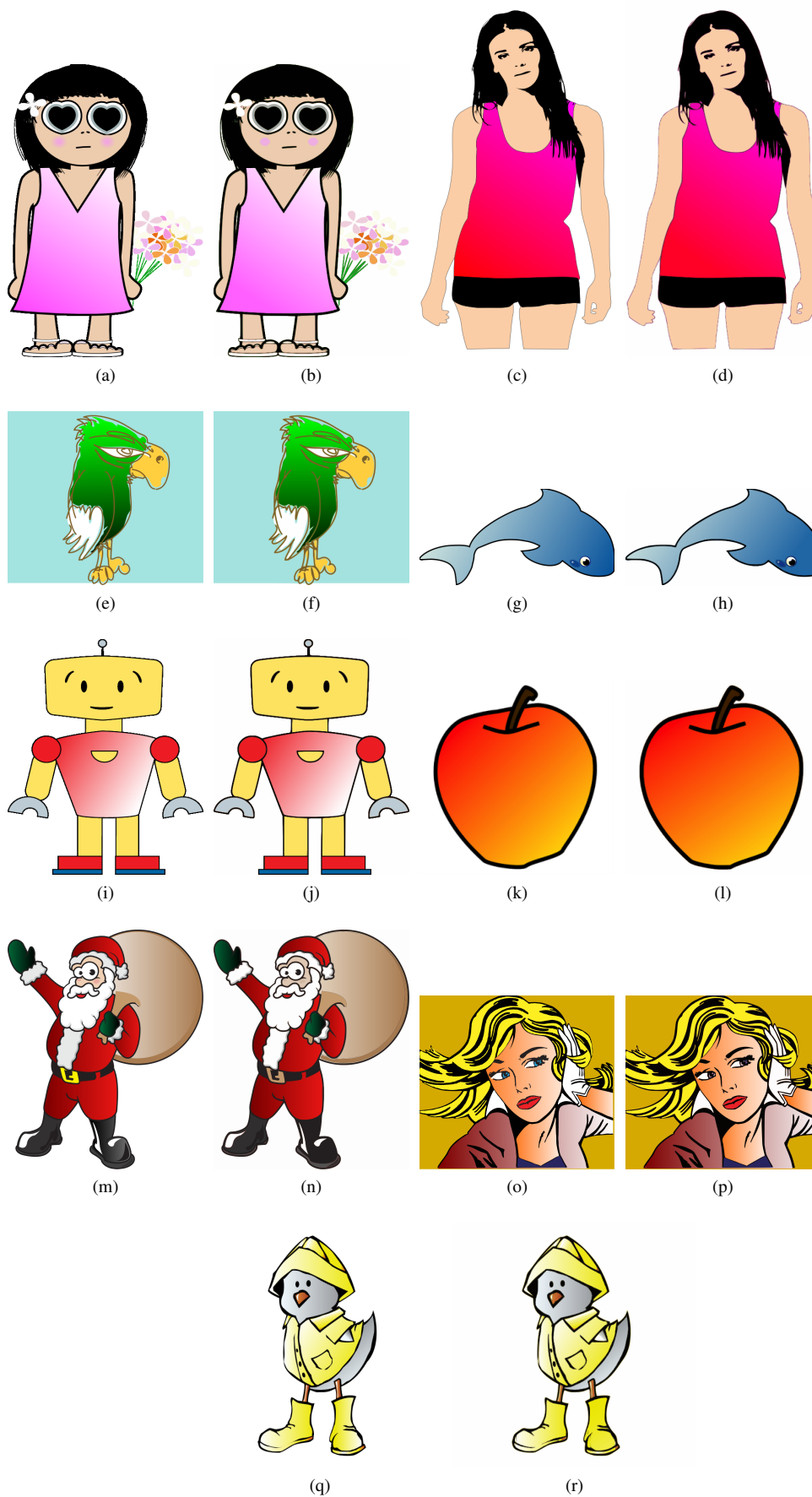
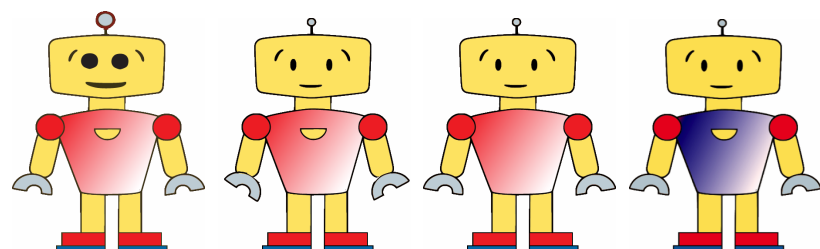
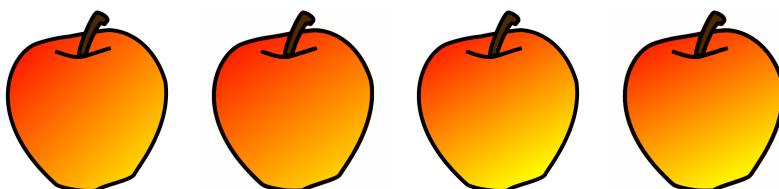


Figure 6: The results with our approach. Original image is on the left and the final vector image is shown on right.

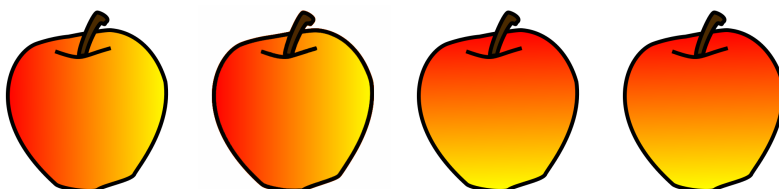


(a) Editing the output vector: Scaled the body parts. (b) Editing the output vector: Rotated the arm levers. (c) Editing the output vector: Removed a path. (d) Editing the output vector: The original linear gradient color stops (as shown in figure 6) were towards red to white. Using Inkscape, we edited the output so that the gradient stops are changed to blue and white.

Figure 7: Editing the final output.

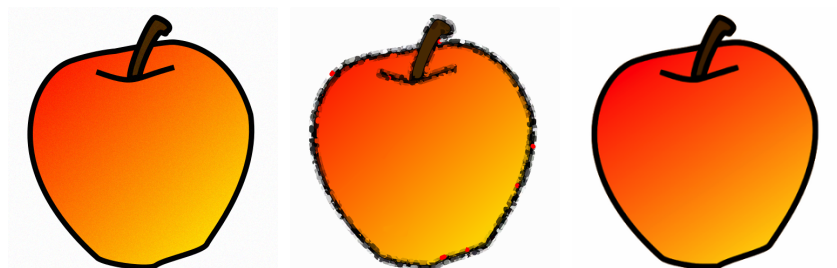


(a) Original gradient direction: 1.0, color varying from (255, 0, 0) to (255, 255, 0). (b) Computed gradient direction: 1.2, color varying from (255, 4, 0) to (255, 248, 0). (c) Original gradient direction: 1.75, color varying from (255, 0, 0) to (255, 255, 0). (d) Computed gradient direction: 1.85, color varying from (255, 8, 0) to (255, 242, 0).



(e) Original gradient direction: 0, color varying from (255, 0, 0) to (255, 255, 0). (f) Computed gradient direction: 0, color varying from (255, 2, 0) to (255, 252, 0). (g) Original gradient direction: ∞ , color varying from (255, 0, 0) to (255, 2, 0). (h) Computed gradient direction: ∞ , color varying from (255, 8, 0) to (255, 251, 0).

Figure 8: Comparison of computed gradient with original known gradient in image.



(a) Input noisy Image. (b) Ardeco output: 1200 small patches. (c) Our output: Only four patches.

Figure 9: Noisy images and comparison with ARDECO.

Improved Interactive Reshaping of Humans in Images

Nidhi Arora

IIT Delhi
India

nidhi@cse.iitd.ac.in

Harsh Kumar

IIT Delhi
India

harsh23031989@gmail.com

Jagjeet S. Dhaliwal

IIT Delhi
India

cs5080212@cse.iitd.ac.in

Prem Kalra

IIT Delhi
India

pkalra@cse.iitd.ac.in

Parag Chaudhuri

IIT Bombay
India

paragc@cse.iitb.ac.in

ABSTRACT

In this paper, we present an interactive and flexible approach for realistic reshaping of human bodies in a single image. For reshaping, a user specifies a set of semantic attributes like weight and height. Then we use a 3D-morphable model based image retouching technique for global reshaping of the human bodies in the image such that they satisfy the semantic constraints specified by the user. We address the problem of deformation of the environment surrounding the human body being reshaped, which produces visible artifacts, especially noticeable at regions with structural features, in prior work. We are able to separate the human figure from the background. This allows us to reshape the figure, while preserving the background. Missing regions in the background are inpainted in a manner that maintains structural details. We also provide a quantitative measure for distortion and compare our results with the prior work.

Keywords

image retouching, image manipulation, warping, deformation, reshaping, inpainting.

1 INTRODUCTION

Professional image editing packages like Adobe Photoshop often limit themselves to local modification of the image whereas retouching tasks such as increasing or decreasing the height or weight of the human bodies in the images require global consistency to be maintained across the human body. This requires professional skills and novice users find it hard to achieve such consistency. Hence, one comes across many such images where the use of Photoshop is quite evident.

Zhou et al. [ZH10] propose an interactive approach based on a 3D-morphable model to deform individual parts of the body to achieve global editing consistency and desired spatially-varying deformation within and across individual body parts. The approach doesn't rely on 3D-reconstruction of the human body from the image but applies a body-aware warping to the image for

reshaping human bodies in a single image. The method comprises of the following steps:

- A morphable 3D-model is roughly matched to the human body in the image by user interaction.
- The 3D-model is deformed by changing the semantic attributes like height and weight.
- The image is then warped such that the human body in the image reflects the changes in the matched 3D-model by following the changes in the 2D-contours of the body with respect to the pose of the human in the image.

The above method is effective for human bodies with a variety of poses, shapes and in presence of loose clothing. It, however, introduces artifacts in the background of the image, noticeable especially in the structural features like walls, floor and other features. Consider, for example Figure 1, where the height of the person is decreased. We can observe that artifacts are introduced in the horizontal structure and the shadow of the person as highlighted in Figure 1(b). We address this problem of deformation of the background of the image by separating the warping of the human body from the background of the image (refer Figure 1(c)). Our solution relies on inpainting of the background image after delineating the human shape that needs a change. We use

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

optical flow as a quantitative measure to obtain the distortion in the background and show that our approach has a lower distortion. The separation of the image into foreground with the human shape and background offers us flexibility to even change the pose of the human body in a limited way.

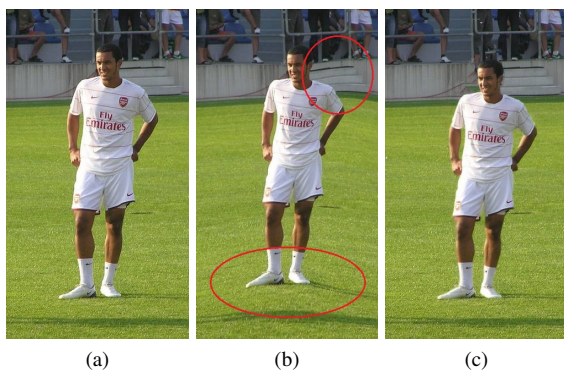


Figure 1: Structural artifacts introduced in the image using [ZH10]; (a) Input image, (b) Resulting image with decreased height and noticeable deformation of the background, (c) Image obtained using our approach.

The paper is organized as follows. In Section 2, some previous research carried out in this area and its allied fields is discussed. Section 3 provides a brief overview of the approach followed by Zhou et al. [ZH10] and highlights its shortcomings. Section 4 describes our proposed approach. Section 5 presents some of the results followed by Section 6 that concludes our work.

2 RELATED WORK

This work is related to a variety of fields. We briefly describe the state-of-the-art techniques for the relevant domains. Since our work is based on the method proposed by Zhou et al. [ZH10], some of the related work cited in that paper is revisited in this section. In addition, we also look at other relevant literature.

2.1 Image Retouching, Warping and Resizing

Retouching images incorporates the use of several image editing tools, for e.g., tone adjustment, recoloring, image composition, image repairing, image denoising, image warping etc. Most existing retouching tools operate at the pixel level and are effective for low level editing tasks [EP08]. However, they are not suitable for high-level editing tasks because they involve user interaction for maintaining the coherence of editing operations [AC02]. Image warping and resizing methods like Moving Least Squares (MLS) [SM06] and Radial Basis Function (RBF) [AR95] propagate the changes from the control handle to the rest of the image. Zhou et al. [ZH10] propose a warping approach for parametric reshaping of humans in images to adhere the human

shape to the specified semantic attributes by resizing the human body along and orthogonal to skeletal bones to achieve a global consistency.

2.2 3D-Morphable Models with Pose Fitting and Shape Selection

SCAPE [AS05] encodes the pose and shape of the human body separately. The pose is stored with the help of an underlying skeleton while the shape of the person is encoded using variational methods or envelope skinning. The outcome of the two steps is combined for the generation of new shapes. Hasler et al. [HS09] introduce a model which encodes both the shape and pose of the model using a translation and rotation invariant encoding. Many automatic pose estimation methods from a single image have been proposed (see [HW09] and references therein). However, it requires a certain amount of user assistance to obtain more reasonable poses ([DA03], [HK07], [CT12], [TM11]). Richter et al. [RV12] present a system for real-time deformation of the shape and appearance of people who are standing in front of a depth+RGB camera, such as the Microsoft Kinect.

In the next section, we give a brief overview of the method proposed by Zhou et al. [ZH10] and outline the limitations of this method.

3 PARAMETRIC RESHAPING OF HUMAN BODIES

Changing the semantic attributes of the human body in the image requires global consistency to be maintained in and across the individual body parts to produce visually pleasing results. A morphable 3D-model of the human body is used to guide the global reshaping of the human body in the image.

The method (refer to Figure 2) requires matching a 3D-morphable model, by adjusting its pose and shape parameters to the human body in the image. The pose and shape parameters of the morphable model are taken as user input for defining the pose and the semantic attributes for the desired change of shape. After matching the 3D-morphable model to the human body in the image, the model is morphed to change the semantic attributes according to the user input. The changes in the fitted model with respect to its 3D-skeleton are used to guide the image warping especially at the projected contours of the fitted model. A body-aware image warping coherently resizes the body parts along the direction parallel and perpendicular to the bone axes of the 3D-skeleton to incorporate the length changes in the corresponding directions. Thus, the human body is parametrically reshaped with the help of user interaction.

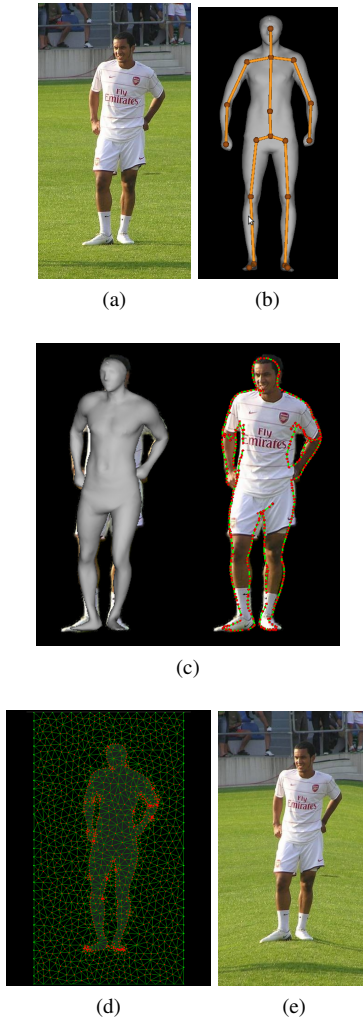


Figure 2: Overview of the parametric reshaping using 3D-morphable model [ZH10]: (a) Original image, (b) mesh rigged with skeleton, (c) pose and shape deformation to roughly match the human body in the image and the image contour, (d) image embedded into 2D-triangular mesh using image subject contour, (e) model is deformed to adhere to new semantic values, changes in the 3D-morphable model are used to drive image warping.

However, we observe that the above approach causes artifacts to the surrounding environment of the human body that is reshaped. For example, in Figure 1, deformation of the background (especially the structural artifacts like stairs and the railings of the stadium and the shadow of the human) is quite noticeable and hence, the resultant image shows visible artifacts when the height of the person is decreased.

Consider Figure 3, the height and weight of Muhammad Ali has been decreased in the resultant image. The unnatural elongation of the hands and the feet of Muhammad Ali as highlighted in the figure occurs be-

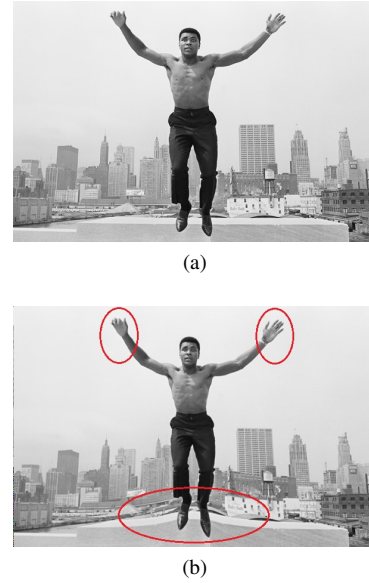


Figure 3: (a) Original image, (b) Unnatural elongation of hands and feet of the person is visible in the output image.

cause of the proximity of the hands and feet to the border of the image which remains unchanged.

In the next section, we address the above mentioned shortcomings and improve this technique.

4 PROPOSED APPROACH

As discussed in the previous section, the above mentioned technique introduces artifacts in the resultant image and hence, the results are not visually pleasing under moderate to severe deformations even though they achieve the required change in the semantic attributes of the human. It is observed that pinning of the image boundaries results in unnatural elongation of the body parts close to the boundaries. Pinning leads to artifacts due to the deformation of triangles near the image boundaries when the human body undergoes deformation according to the changes in the 3D-morphable model. One can pad the image while keeping the vertices on the boundary of the padded image fixed or pad the image and relax the constraints at the boundary of the padded image. This, however, gives only limited improvement.

We observe that the body-aware warping of the whole image results in structural artifacts as the changes in the human body are propagated to the background. These structural artifacts can be removed by separating the warping of the human body from the background of the image by using the proposed technique consisting of the following steps:

- The human bodies, to be warped, are segmented from the image as foreground using GrabCut [RK04].

- The hard segmentation, thus, obtained is used as input for extracting the foreground color, background color and alpha value for each pixel of the image.
- Body-aware image warping, based on the parametric deformation of the 3D-morphable model, is then applied to the foreground image and the alpha matte.
- The hole generated in the background is inpainted using a suitable technique.
- The resulting image is then, generated by combining the warped foreground image and the inpainted background using the warped alpha values for each pixel.

4.1 Segmentation of the Human Body

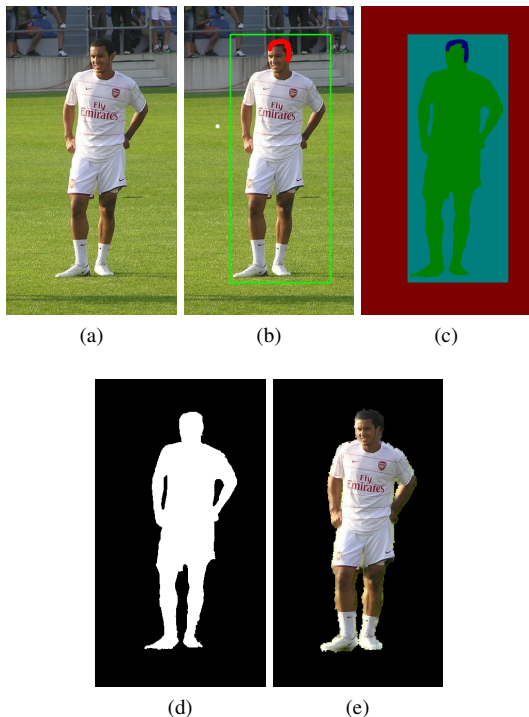


Figure 4: Segmentation of the human body from the image using GrabCut: (a) Input image, (b) Scribbles provided to drive the segmentation, (c) Regions formed by GrabCut: red, blue, green and dark blue represent background, probable background, probable foreground and foreground respectively, (d) Hard segmentation mask obtained by combining foreground and probable foreground regions, and (e) Human body segmented from the image using hard segmentation mask.

The human body to be warped is segmented from the image using GrabCut [RK04] where the user interaction is limited to drawing a rectangle to mark the background and probable foreground area. GrabCut [RK04] uses an iterative minimization graph-cut algorithm using Gaussian Mixture Models to segment the

foreground and reduces the amount of the user interaction. If the results are not approximately correct, more information is provided using scribbles, marking the areas of the image as background and foreground and running the iterative mechanism once again till the segmentation of the human body from the image is approximately correct. The foreground scribbles and the probable foreground are then used to form the hard segmentation of the human body that is to be warped (refer Figure 4). Many other methods exist for segmenting the desired foreground from the image. Magic Wand tool uses the user input to form color statistics from the specified region and computes a connecting region of pixels such that the selected pixels fall within some adjustable tolerance of the color statistics. We have found GrabCut [RK04] suitable for our purpose.

4.2 Image Matting

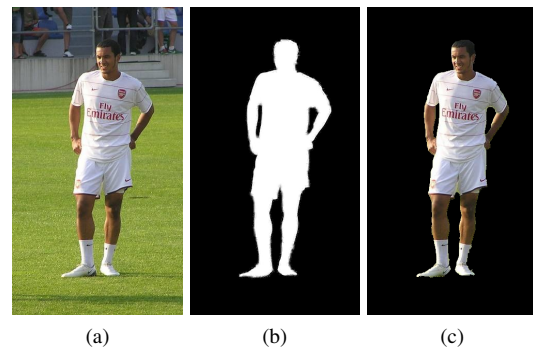


Figure 5: (a) Input image, (b) Alpha matte, (c) Foreground color image.

In our approach, hard segmentation obtained from segmenting the foreground object from the complex background is transformed into a trimap by using erosion on the user input (scribbles) to drive the calculation for the three components in a narrow strip around the hard boundary (refer Figure 5). Thus, we separate the foreground color, background color and alpha value for each pixel of the image. There exist a number of methods for matting [CS01], [SJ04], [LW06], we have adopted a simple approach and found it adequate for our purpose.

4.3 Image Warping

To warp the image, we follow the same method as proposed by Zhou et al. [ZH10] as described in the previous section. For pose matching, we have found that the data of 3D-morphable models need axis alignment to the coordinate system induced by the image. The mesh needs to be transformed so that it is upright and facing the user. To achieve the desired orientation of the mesh, it is rotated to align the plane of symmetry to the global YZ-plane. To compute the plane of symmetry of the human mesh placed arbitrarily in the space, the original

mesh is registered to the triangular mesh obtained by reflecting the original mesh (termed as mirrored mesh) about any arbitrary plane in the space. The original mesh and the mirrored mesh are roughly aligned to each other by performing a coarse registration using the optimal set of Persistent Feature Histograms [AR95]. The axes of the minimum area bounding box of the vertices projected on the plane of symmetry are aligned to the world coordinate axes. For fine alignment of the mesh obtained so far, it is registered against the template mesh using a corresponding point set registration. Now, the mesh obtained from the database is upright and facing the user.



Figure 6: Sample mesh generated from the database using given semantic attributes and the required orientation of the mesh.

4.4 Background Filling

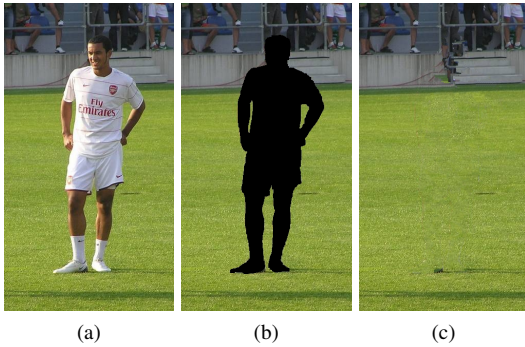


Figure 7: (a) Input image, (b) Target region for inpainting, (c) Inpainted background image.

Background completion is performed by using an image inpainting technique [CP04]. The target region is specified by creating a mask using alpha values obtained in the previous step. A threshold is chosen on the alpha values such that all the pixels with alpha value more than the threshold are taken to be foreground and hence are removed from the image and form the target region to be inpainted as shown in Figure 7(b). Arora et al. [AK12] propose a method to improve exemplar based inpainting [CP04] by tweaking the values of various parameters like patch size, shape and size of the mask. They use the technique of inpainting for filling

in the cracks and restoring old paintings. We have used their approach [AK12] for background completion.

4.5 Remaining Steps

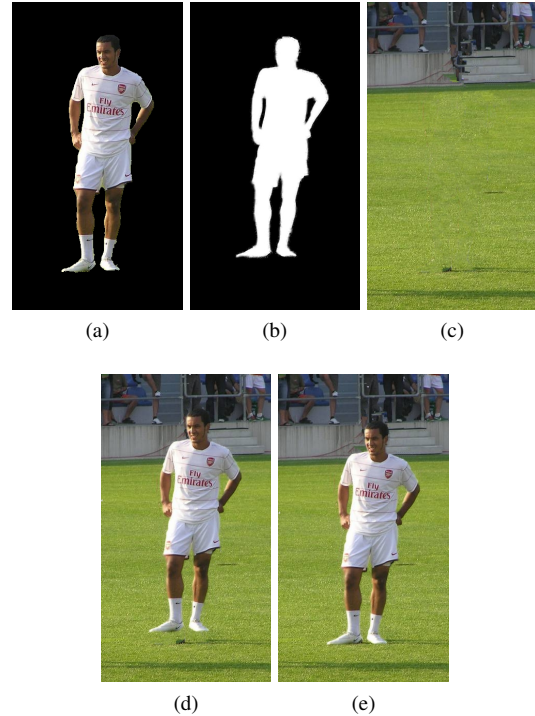


Figure 8: Combining the warped foreground color image and the inpainted background to form the resultant image; (a) Warped foreground color image, (b) Warped alpha matte, (c) Inpainted background color image, (d) Combining the above without any translation of foreground with respect to the background, and (e) Combining the above after translating the foreground image for a more visually-pleasing result.

Thus, we obtain the alpha matte, the foreground image and the inpainted background image by following the above mentioned steps. The 3D-morphable model is roughly matched with the human body in the blended foreground image (foreground image combined with the alpha matte). The padded blended foreground image is embedded into a 2D-triangular mesh using the image contour and the boundaries of the padded image and the changes in the 3D-morphable model (conforming to the new semantic attribute values) are then used to warp the 2D-triangular mesh to obtain the new 2D-triangular mesh. The new 2D-triangular mesh is used to warp the foreground image and the alpha matte of the segmented human body. The foreground image and the inpainted background image are then blended together to form the resultant image using the alpha matte. The foreground image and the alpha matte sometimes need to be translated to produce more-visually pleasing effects. For example, in Figure 8 the foreground image

and the alpha matte need to be translated so that the shadow and the human are in contact with each other and hence, produce a more-visually pleasing and plausible result.

5 RESULTS

Our proposed method is applied on a variety of images of human subjects with various poses and shapes. The human subjects in the image are morphed to reflect the changes in the semantic attributes as desired by the user. Figure 10, Figure 11 and Figure 12 shows the comparison of the results from [ZH10] and our approach. Artifacts are clearly visible in the regions with structural features as highlighted in the images. The environment surrounding the human body being reshaped is noticeably deformed in Zhou et al. [ZH10] results. Our results do not suffer from such problems because of the use of inpainting in our pipeline. In Figure 12, we have also elongated the shadow of the girl along with increasing her height.

We also compare the results in terms of optical flow field, a quantitative measure of distortion. Another measure of distortion is Gradient Vector Flow (GVF) [SB12] but we have used optical flow field for evaluation. In Figure 9, we compare the optical flow field obtained from Zhou et al. result and our result for Figure 1. Figure 9(a) and Figure 9(c) are the color coded output for optical flow. Color coding represents the magnitude of flow field. The range of the colors is from yellow to violet where violet represents large magnitude of the flow and yellow color means a lower value of flow field. Figure 9(b) and Figure 9(d) are the corresponding pictorial representation in the form of arrow diagrams. Table 1 denotes the quantitative measure of the magnitude of optical flow for both the results. Total flow value is the sum total of the magnitude of the optical flow of all the pixels of the image, then we divide it by the total number of pixels in the image to obtain the per-pixel optical flow value. To calculate the flow values for the background pixels in the image, mask is used to separate foreground from background. Same mask is also used to compute the background in Zhou result to have uniformity of comparison. In our result, both the flow value per pixel and the flow value for background is lower than the Zhou et al. result which means that our result has lower distortion and specifically, lower distortion of the background. Similarly, in Figure 10 and Figure 11, comparison in terms of optical flow fields is also shown. Table 2 represents the quantitative comparison of the per-pixel flow values for Fig 10 and Fig 11. Flow values for our results are much lower than Zhou et al. results.

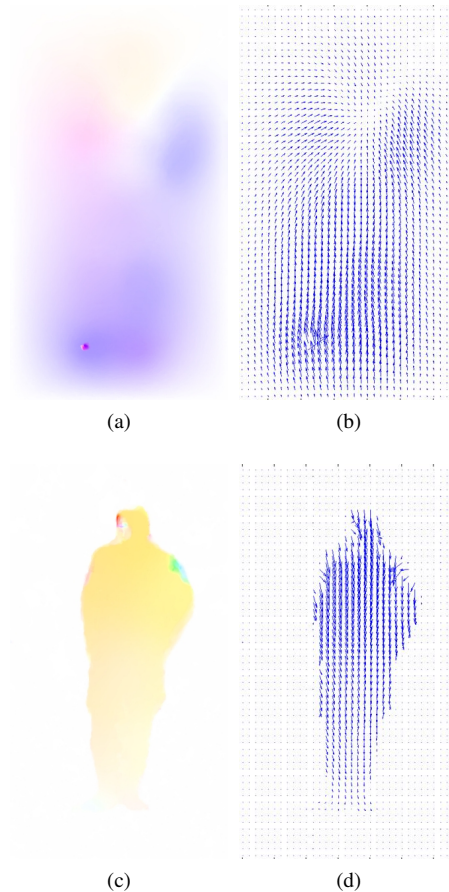


Figure 9: Optical flow field obtained for Fig 1: (a), (b) Flow field obtained from Zhou et al. result and (c), (d) Flow field obtained from our result.

Results	Flow value per pixel	Flow value for background(per-pixel)
Zhou et al. result	12.3911	12.8948
Our result	4.1860	0.7214

Table 1: Optical flow values for Fig 9 (Size: 600×331).

5.1 Human Subject with Loose Clothing

We have handled the case of loose clothing in the same way as Zhou et al. [ZH10] have handled. We create a larger mask covering the entire clothing of the person for inpainting the background. Pose fitting and shape selection are performed in the same way as explained in our pipeline. In Figure 13, waist girth of the girl has been decreased.

5.2 Multiple Human Subjects

We address another application in which an image comprise of multiple human subjects. Our objective is to change the semantic attributes of both the subjects. This is done in two passes. We create three masks, two for the individual humans and a combined mask covering both the subjects. We change the semantic attributes of

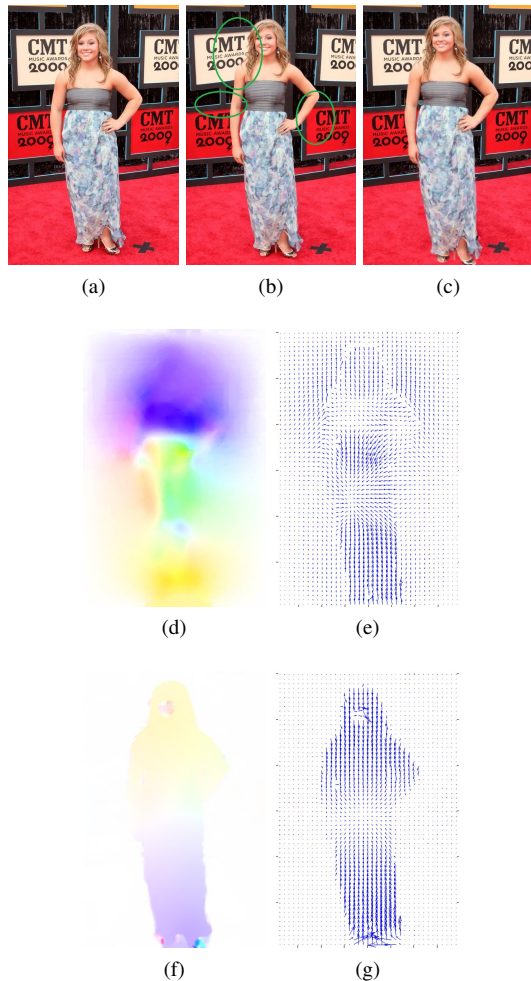


Figure 10: Comparison of results from Zhou et al. [ZH10] and our approach: (a) Input image, (b) Result using [ZH10] with height increased, (c) Result from our approach, (d), (e) Optical flow field obtained from Zhou et al. result and (f), (g) Optical flow field obtained from our result.

both the subjects individually following the above mentioned pipeline. The mask covering both the humans is used for inpainting the background. We then, paste the reshaped subjects onto this inpainted background. For instance, in Figure 14, height of both the subjects is decreased and in Figure 15, weight is decreased and height is increased of both the subjects.

5.3 Change of Background Setting

We can also change the semantic attributes of a person and place it in a different setting altogether. According to our approach, we segment out the foreground from the background and parametrically reshape the foreground, which basically contains the human body. We then, composite this reshaped subject onto a different background. In Figure 16, we change the height of the character in Figure 16(a) and place her in a different

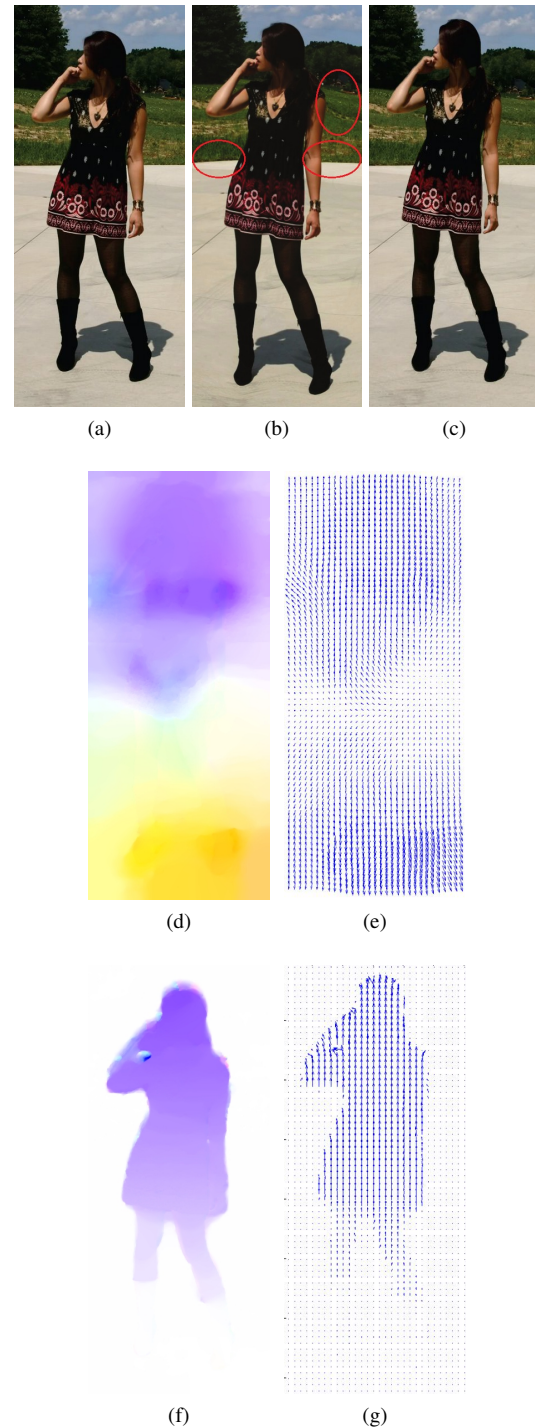


Figure 11: (a) Original image, (b) Result using [ZH10] with increased height of the human subject and (c) Resulting image with increased height from our approach, (d), (e) Optical flow field obtained from Zhou et al. result and (f), (g) Optical flow field obtained from our result.

background setting in Figure 16(b). Shadow is cast in the resultant image assuming the position of the light source and is then composited with the background.

Figures	Zhou et al. results(Flow value per pixel)	Our results(Flow value per pixel)
Fig.10(600×393)	7.6880	4.5066
Fig.11(726×307)	10.8101	3.8052

Table 2: Optical flow values for Zhou et al. results and our results for Fig 10 and Fig 11.

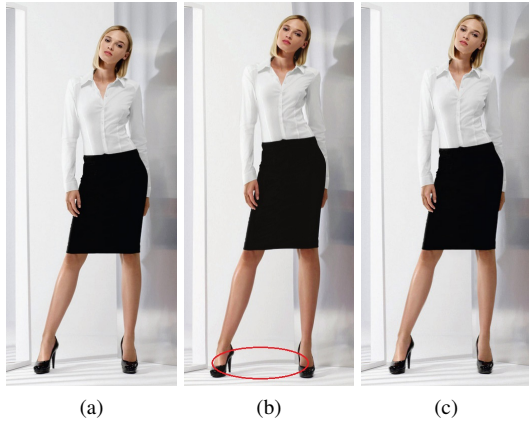


Figure 12: (a) Original image, (b) Result using [ZH10] with increased height of the human subject and (c) Resulting image with increased height from our approach.

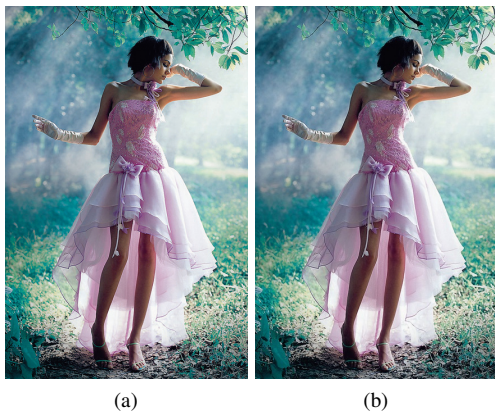


Figure 13: (a) Original image and (b) Resulting image with decreased waist girth of the human subject.

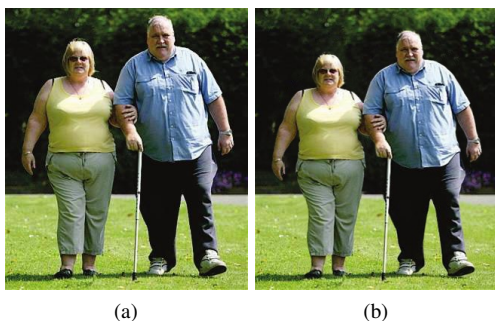


Figure 14: (a) Original image and (b) Resulting image with decreased height of both the human subjects.



Figure 15: (a) Original image, (b) Resulting image with decreased weight and increased height of both the human subjects.

Since the method proposed by Zhou et al. [ZH10] does not separate the foreground and background, so, it cannot be used to change the background setting of a human subject.



Figure 16: (a) Original image, (b) Human subject placed in a different background setting.

6 CONCLUSION AND FUTURE WORK

This paper presents an interactive and flexible approach for realistic reshaping of human bodies in an image. We perform parametric reshaping using a 3D-morphable model to achieve globally consistent manipulation effects. A user specifies a set of semantic attributes like weight, height and others as proposed earlier in 3D-morphable model based image retouching technique [ZH10] for global reshaping of human bodies in an image. We address the problem of deformation of the background of the image because of the propagation of the retouching effects to the background. We follow the approach in which we separate the foreground and background. Foreground is reshaped and background is inpainted maintaining the necessary structural details.

The main contribution of this paper is to combine set of techniques to obtain improved results. The improve-

ment is shown both in terms of visual results and a quantitative measure.

As a future work, we are interested in extending our approach for reshaping subjects in videos.

7 ACKNOWLEDGEMENTS

We would like to thank Nils Hasler [HS09] for providing the database of scanned human meshes. We would like to thank Shizhe Zhou [ZH10] and Ligang Liu [ZH10] for the use of some of the images that we use as input to test our methods. We have taken these images from Zhou et al. [ZH10] paper and their supplementary materials available online.

8 REFERENCES

- [AC02] W. A. Barrett and A. S. Cheney, "Object-based image editing," *ACM Transaction on Graphics, Proceedings of SIGGRAPH 2002*, vol. 21, issue. 3, pp. 777-784, July 2002.
- [AS05] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers and J. Davis, "SCAPE: shape completion and animation of people," *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2005*, vol. 24, issue. 3, pp. 408-416, July 2005.
- [AR95] N. Arad and D. Reisfeld, "Image warping using few anchor points and radial functions," *Computer Graphics Forum*, vol. 14, pp. 35-46, 1995.
- [AK12] N. Arora, A. Kumar and P. Kalra, "Digital restoration of old paintings," *WSCG International Conference on Computer Graphics, Visualization and Computer Vision 2012*, pp. 347-356, June 2012.
- [CT12] T. Chen, P. Tan, L. Q. Ma, M. M. Cheng, A. Shamir and S. M. Hu, "Poseshop: Human image database construction and personalized content synthesis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, issue. 5, pp. 824-837, May 2013.
- [CS01] Y. Chuang, B. Curless, D. Salesin, and R. Szeliski, "A bayesian approach to digital matting," in *proceedings of IEEE CVPR 2001*, vol. 2, pp. 264-271, Dec. 2001.
- [CP04] A. Criminisi, P. Pérez and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," *IEEE Transactions on Image Processing*, vol. 13, issue. 9, pp. 1200-1212, Sept. 2004.
- [DA03] J. Davis, M. Agrawala, E. Chuang, Z. Popovic and D. Salesin, "A sketching interface for articulated figure animation," *SCA '03*, pp. 320-328, 2003.
- [EP08] K. Eismann and W. Palmer, "Adobe photoshop restoration and retouching," *New Riders Press, 3rd edition*.
- [HS09] N. Hasler, C. Stoll, M. Sunkel, B. Rosenhahn and H. P. Seidel, "A statistical model of human pose and body shape," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 28, issue. 2, pp. 337-346, March 2009.
- [HK07] A. Hornung, E. Dekkers and L. Kobbelt, "Character animation from 2D pictures and 3D motion data," *ACM Transactions on Graphics*, vol. 26, issue. 1, Jan. 2007.
- [HW09] Z. Hu, G. Wang, X. Lin and H. Yan, "Recovery of upper body poses in static images based on joints detection," *Pattern Recognition Letters*, vol. 30, issue. 5, pp. 503-512, Apr. 2009.
- [LW06] A. Levin, D. Lischinski, and Y. Weiss, "A closed form solution to natural image matting," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR '06*, vol. 1, pp. 61-68, June 2006.
- [RV12] M. Richter, K. Varanasi, N. Hasler and C. Theobalt, "Real-time reshaping of humans," *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 340-347, 2012.
- [RK04] C. Rother, V. Kolmogorov and A. Blake, "GrabCut: interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2004*, vol. 23, issue. 3, pp. 309-314, Aug. 2004.
- [SM06] S. Schaefer, T. McPhail and J. Warren, "Image deformation using moving least squares," *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2006*, vol. 25, issue. 3, pp. 533-540, July 2006.
- [SJ04] J. Sun, J. Jia, C. K. Tang and H. Y. Shum, "Poisson matting," *ACM Transaction on Graphics, Proceedings of SIGGRAPH 2004*, vol. 23, issue. 3, pp. 315-321, Aug. 2004.
- [TM11] M. T. Islam, K. M. Nahiduzzaman, Y. P. Why and G. Ashraf, "Informed Character pose and proportion design," *The Visual Computer: International Journal of Computer Graphics - Special Issue on CYBERWORLDS 2010*, vol. 27, issue. 4, pp. 251-261, Apr. 2011.
- [ZH10] S. Zhou, H. Fu, L. Liu, D. Cohen-Or and X. Han, "Parametric reshaping of human bodies in images," *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2010*, vol. 29, issue. 4, July 2010.
- [SB12] S. Battiato, G. M. Farinella, G. Puglisi, D. Raví, "Content-aware image resizing with seam selection based on gradient vector flow," in *proceedings of IEEE International Conference on Image Processing ICIP 2012*, pp. 2117-2120, Sept. 2012.

Similarity Detection for Free-Form Parametric Models

Quoc-Viet Dang
University of Toulouse,
France
qdang2@n7.fr

Sandrine Mouysset
University of Toulouse,
France
sandrine.mouysset@irit.fr

Géraldine Morin
University of Toulouse,
France
morin@n7.fr

Abstract

In this article, we propose a framework for detecting local similarities in free-form parametric models, in particular on B-Splines or NURBS based B-reps: patches similar up to an approximated isometry are identified. Many recent articles have tackled similarity detection on 3D objects, in particular on 3D meshes. The parametric B-splines, or NURBS models are standard in the CAD (Computer Aided Design) industry, and similarity detection opens the door to interesting applications in this domain, such as model editing, objects comparison or efficient coding. Our contributions are twofold: we adapt the current technique called votes transformation space for parametric surfaces and we improve the identification of isometries. First, an orientation technique independent of the parameterization permits to identify direct versus indirect transformations. Second, the validation step is generalized to extend to the whole B-rep. Then, by classifying the isometries according to their fixed points, we simplify the clustering step. We also apply an unsupervised spectral clustering method which improves the results but also automatically estimates the number of clusters.

Keywords

similarity detection, parametric surfaces, isometry, spectral clustering

1 INTRODUCTION

Parametric surfaces, in particular Non-Uniform Rational B-Spline (NURBS), provide a powerful tool in the hands of the academic and industrial communities concerned with the design and analysis of objects [Dim99a]. NURBS based B-reps (Boundary representations) are industrial standards and are widely used in different domains such as molecular chemistry [Baj97a], 3D geographical information systems [Cau03a] and mechanical components design [Chu06a]. Additionally, similarity within a 3D shape is a common phenomenon both in natural and in synthetic objects. Many objects are composed by similar parts up to a rotation, a translation or a reflection. Geometric redundancy is an essential property that artists must strive with in their works, that 3D designers must provide in their conceptions so that the human vision system perceives the object beauty. Similarity detection within 3D models is then a first step towards numerous interesting applications. In CAD, automatic search of similarity between CAD models is used primarily for model retrieval and indexing in large scale CAD

databases [Car06a, Chu06a, Che12a, Liu13a]. In that context, end-users request automatic searches for "similar enough" designs according to a given model or sketch. Thus, the design reuse is encouraged by making use of existing components. For 3D meshes, many applications are studied such as pattern recognition, form editing or data completion. For example, Mitra et al. presented a symmetrization algorithm for geometric objects that enhances approximate symmetries of a model while minimally altering its shape [Mit07a]. Chaouch et al. [Cha08a] considered the reflection as the main characteristic to align their 3D models. Li et al. [Li11a] proposed a skull completion framework based on symmetry and surface matching. With the particular attractiveness of NURBS surfaces in 3D design industry, the similarity detection would certainly be useful. In fact, designers rarely start their works from scratch, but rather adapt existing models to meet new requirements. Statistically, it is shown that more than 75% of design activity involves reusing existing designs or starting from existing designs to address new designs [Iye05a]. Besides, parametric NURBS representations allow to easily and reliably access differential informations over the surfaces. Their representation by control points also gives the designer intuitive control. Hence, local similarities detection should be interesting for *reverse engineering*, allowing in one hand the analysis of a given 3D model, and in the other hand shape editing that is coherent with the detected similarities. Data compression in order to limit the storage size of a model can also benefit from

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

the redundancy identified in similar parts. As far as we know, no research so far was dedicated to detecting the local similarities on parametric models like B-Spline or NURBS based B-reps. This article presents a method allowing the identification of NURBS surface patches that are similar to an approximated isometry. Our contributions are as follow. First, to find the best orientation of vectors of the characterized local frame at a point on the surface, we propose a simple method by analysing neighbourhood properties. We thus distinguish between direct and indirect isometries and propose to partition the isometries into five subsets. This classification simplifies the clustering and improves the identification of isometries. We further improve the clustering step by applying a spectral clustering algorithm. Unlike Mean Shift algorithm, our approach is fully unsupervised, and as such, is able to group automatically clusters without customizing global parameters. The remainder of this article is organized as follow. Section 2 reviews some previous works and our approach in this work. Section 3 describes the proposed pipeline of our algorithm that is detailed in the following sections. Section 8 shows some results of similarity detection among numerous experiments. Section 9 presents our conclusion and future works.

2 PREVIOUS WORK

In recent years, many articles have been published on similarity detection both in 2D image processing and in 3D modeling. In a first approach, Zabrodsky et al. [Zab95a] quantified existing symmetries within 2D and 3D objects, using a metric called the *symmetry distance*. The symmetry distance of a shape is defined to be the minimum mean squared distance required to move points of the original shape in order to obtain a symmetrical shape. Sun et al. [Sun97a] converted the symmetry detection problem into the correlation of Gaussian images; rotational and bilateral symmetries are identified by applying orientation histograms.

For 3D shape matching, two dominant techniques were proposed. First, global feature-based techniques represents 3D objects as a set of global features, for example, spherical-kernel moments [Cyb97a], or reduced feature vectors [Car06a]. The other set of methods uses graph-based techniques: the solid models are converted into attributed graphs that represent the geometrical and topological relationship between models entities [Hil01a, Ma10a]. However, in both cases, these techniques can neither identify similar parts within a model nor compute the transformation between these similar parts. Recently, many papers proposed to identify similarities within 3D meshes [Kaz04a, Pod06a, Ber08a, Bok09a, Lip09a, Mit13a] with different approaches like planar-reflective symmetry, graph-based

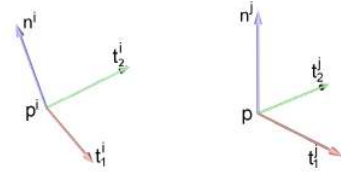


Figure 1: Local Frames of two similar points p_i et p_j according to right hand rule.

matching, or votes transformation space. Kazhdan et al. [Kaz04a] introduced a *reflective symmetry descriptor* that represents a measure of reflective symmetry for an arbitrary 3D model for all planes through the model's center of mass. Podolak et al. [Pod06a] generalized this approach to identify symmetries of 3D objects associated with an arbitrary plane. Graph-based approach requires detecting local features on 3D shape from which a neighborhood graph is build to describe the coarse scale similarity structure of the object. Berner et al. [Ber08a] perform subgraph matching in graphs of feature points while Bokeloh et al. [Bok09a] apply feature lines.

Other recent works [Lip09a, Mit13a] applied new technique in symmetry detection that we call *votes transformation space*. This technique bears some similarity to the Hough transform: points on the model with similar features are paired. A points pair corresponds to the transformation between the two points and their features; these transformations are cast to the transformation space and form a constellation of transformation votes. Clusters of these votes are candidates for defining similar parts in the model. While Mitra et al. [Mit13a] use Euclidean transformations as the feature to extract similarity, Lipman et al. [Lip09a] adopt Möbius transformations.

Among these approaches, the votes transformation space attracts our interest since it allows to retrieve a large class of potential transformations and it is able to identify similar parts in existing 3D objects and to characterize the transformation. In order to give a general view of this scheme, we detail the algorithm proposed in [Mit13a] that consists in the following four steps:

1. *Sampling and analysis*: a set of points is sampled over the surface of a 3D object. Since point positions are not sufficient to determine a general Euclidean transformation, geometry features at each sample are computed (the principal curvatures and a local frame composed of the principal directions and a normal vector). The signature is the couple of principal curvatures; points on the surface are paired if they have the same signature.
2. *Pairing*: each pair of points is associated a transformation corresponding to a vote in transformation space. Given two points p_i and p_j with their local

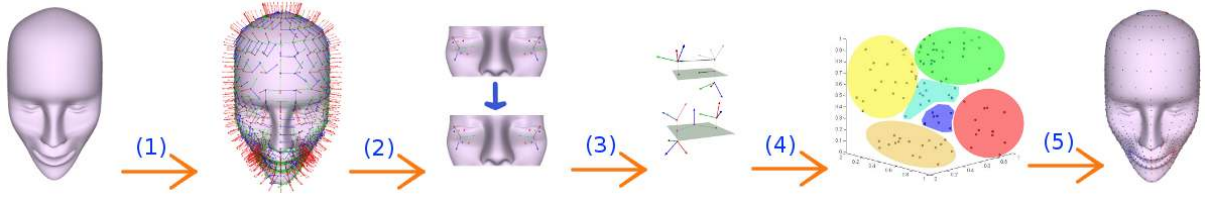


Figure 2: Proposed pipeline – (1) Sampling and signature computation, (2) Pairing and orientation, (3) Classification of isometry, (4) Clustering, (5) Validation.

(orthonormal) frames consisting in two tangents and a normal (figure 1), the transformation T_{ij} is computed so that p_i and its frame are mapped into p_j and p_j 's frame. This transformation is then cast into votes of transformation space Γ .

3. *Clustering*: in transformation space Γ , each point T_{ij} represents a transformation between two similar points. Hence, clusters of similar transformations are identified since they may characterize two similar parts of the object.
4. *Patching*: ideally, a cluster of the previous step is a set of point pairs which belong to a couple of surface patches similar up to a transformation close to the cluster. However, spatial coherence between point pairs is lost in transformation space. Thus, this step enforces spatial coherence of the point pairs by applying an incremental region growing algorithm.

Our proposed pipeline follows the same votes transformation space approach. Our contributions are as follows. First, to find the best vectors orientations of the characterized local frame at a point on the surface, we propose a simple method by analysing neighbourhood properties. We thus distinguish between direct and indirect isometries and propose a partition the isometries into five subsets. This classification simplifies the clustering and improves the identification of isometries. We further improve the clustering step by applying a spectral clustering algorithm. Unlike Mean Shift algorithm, our approach is fully unsupervised, and as such, is able to group automatically clusters without customizing global parameters. In the following section, we described our isometry detection relative to these four steps.

3 PROPOSED PIPELINE FOR ISOMETRY DETECTION

Our work aims at identifying surface patches in a B-rep model that are similar up to an approximated isometry (we do not consider scaling). To identify the similarities, we adapt the *votes transformation space* that are used successfully in 3D meshes area [Lip09a, Mit13a]. Our pipeline consists in five consecutive steps. First, points are sampled over all B-reps of a CAD model by a sampling technique that adapts the parameterization (section 4.3). When the signature at each point is computed, vector directions are determined by parameteri-

zation, so it is not a geometric property of the surface. For this reason, local frames are not coherent, in particular to identify indirect isometries. We propose then a simple method to overcome this problem (section 4.4). Isometries between pairs of points are computed and partitioned into five types, based on orientation and on their fixed points (section 5). Next, clustering is applied in these five different spaces using a fully unsupervised spectral clustering algorithm to extract the evidence of existing similarity in the model (parameters are automatically computed). The isometries classification has two advantages: first it simplifies the clustering, but it also maps the pairs in transformation spaces of reduced dimensions. In this pipeline, the computation of the transformations is a major concern that affects considerably the quality of the result. By parametrizing the isometries differently, we improves the identification of isometries. Finally, similarities among local patches are identified following an adaptive growing process adapted for multiple faces in B-rep models (section 7).

4 COMPUTATION OF THE SIGNATURES

In our setting, we work with B-rep models based on trimmed free-form patches made of NURBS tensor product surfaces. For the first three steps of the similarity detection pipeline, it is sufficient to consider the patches independently. Thus, in this section, we focus on NURBS tensor product surfaces and in particular in computing a set of sample points and their characterizing signatures.

4.1 NURBS based models

Let S be a tensor product NURBS surface of bi-degree (p, q) associated to two knots vectors $\mathbf{u} = \{u_0, \dots, u_n\}$ and $\mathbf{v} = \{v_0, \dots, v_m\}$ and a set of control points $C = \{P_{ij} \mid i \in [0, n-p], j \in [0, m-q]\}$ weighted by $w_{ij} \in \mathbb{R}$, defined by the following equation:

$$S(u, v) = \frac{\sum_{i=0}^{n-p} \sum_{j=0}^{m-q} N_{i,p}(u) N_{j,q}(v) w_{ij} P_{ij}}{\sum_{i=0}^{n-p} \sum_{j=0}^{m-q} N_{i,p}(u) N_{j,q}(v) w_{ij}}. \quad (1)$$

In a B-rep model, faces are not only represented by this type of NURBS, but also by other types such as planes, cylinders or spheres. However, one of the advantages

of NURBS is that we can represent free-form as well as quadric surfaces [Cui11a].

4.2 Local differential properties: computation of the signature

Any point on the parametric surface, corresponding to a parametric coordinates (u, v) , is attached to a set of persistent properties which is called *the signature* at that point. In our work, the signature at each point is composed of the two principal curvatures and an *orthonormal affine frame* having origin at that point, the unit vectors are the normal vector and the two principal directions (i.e. tangent vectors associated to the considered principal curvatures). The signature computation at a specific point on NURBS surface is based on local differential properties that could be evaluated from *the first and the second fundamental form* [Str61a, Far92a]. The *first fundamental form* that describes completely the metric properties of a surface, is defined as the distance of two points on a curve of the surface:

$$ds^2 = E du^2 + 2F du dv + G dv^2 \quad (2)$$

where $E = S_u \cdot S_u$, $F = S_u \cdot S_v$, $G = S_v \cdot S_v$, and ds is also called *the element of arc*.

The *first fundamental form* states that, for a given point p , partial derivatives S_u and S_v generate a tangent plane to the surface of origin p . Hence, the unitary normal vector is:

$$n = \frac{S_u \wedge S_v}{\|S_u \wedge S_v\|} = \frac{1}{\sqrt{EG - F^2}} (S_u \wedge S_v) \quad (3)$$

It associates to non normalized vectors S_u , S_v to form an affine frame of origin p .

Next, *the second fundamental form* of a parametric surface is defined by:

$$\kappa \cos \phi ds^2 = L du^2 + 2M du dv + N dv^2 \quad (4)$$

where $L = S_{uu} \cdot n$, $M = S_{uv} \cdot n$, $N = S_{vv} \cdot n$, and S_{uu} , S_{uv} , S_{vv} are second partial derivatives at p .

Equation (4) means that, for a given direction du/dv in u, v plane and a given angle ϕ , the *first and second fundamental forms* allow us to compute the curvature κ of a curve traced on the surface, also the tangent pointing toward this direction.

For this reason, two symmetric matrices are introduced:

$$\mathcal{F}_1 = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \text{ and } \mathcal{F}_2 = \begin{pmatrix} L & M \\ M & N \end{pmatrix} \quad (5)$$

Because S_u and S_v are linearly independent, \mathcal{F}_1 is always invertible. The matrix $\mathcal{F}_1^{-1} \mathcal{F}_2$ is also symmetric and so always has real eigenvalues and orthogonal eigenvectors. As a result, the two eigenvalues κ_1 , κ_2 are the two *principal curvatures* and the two eigenvectors $t_1 = (\xi_1, \eta_1)^T$, $t_2 = (\xi_2, \eta_2)^T$ define the two *principal directions*:

$$\begin{aligned} t_1 &= \xi_1 S_u + \eta_1 S_v \\ t_2 &= \xi_2 S_u + \eta_2 S_v \end{aligned} \quad (6)$$

As for umbilical points ($\kappa_1 = \kappa_2$), principal directions are not uniquely defined, thus we do not consider them.

For other points, the orientation of t_1 and t_2 depends on the parameterization. Section 4.4 details the way we orient the frame vectors.

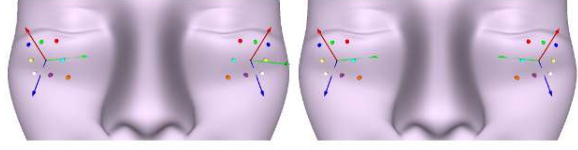


Figure 3: On the left: the orientation of the frame vectors follows the parameterization, so the two frames are not symmetric. On the right: we propose to find a coherent orientation of the vector frames by analyzing the points neighbors. Now, the two frames are symmetric, as is the underlying surface.

4.3 Sampling

Every point on the surface that is associated to a signature characterized by its local differential properties, might be potentially sampled for later computations. By benefiting from the facilities offered by parametric surfaces, a net of sample points on the surface is obtained by sampling uniformly the two parameters u and v (see equation 1). However, the parameterizations between surfaces in B-rep models vary. The uniform sampling along u and v may lead to a sparse net of sample points (figure 4a). To have a relatively uniform distance between points among all surfaces, we propose an iterator method to determine the two parameter gaps based on the distance between two points on each surface (figure 4b). In addition, the sampling affects the following steps of the algorithm in two ways. First, the denser sampling is, the better result is. Second, the denser samples also worsen the performance. For this reason, we evaluate a net of points uniformly on the surface but select randomly a limited number of samples following a uniform law on this points net (figure 4c). Moreover, the initial samples net is reserved for the validation step.

4.4 Robust surface orientation

Two sample points p_i and p_j are considered similar if their principal curvature matches, that is, $\kappa_1^i \sim \kappa_1^j$ and $\kappa_2^i \sim \kappa_2^j$. Two similar points are paired to evaluate the transformation between them. As mentioned in section 3, the orientation of the local frames vectors depend on parameterization. However, a coherent orien-

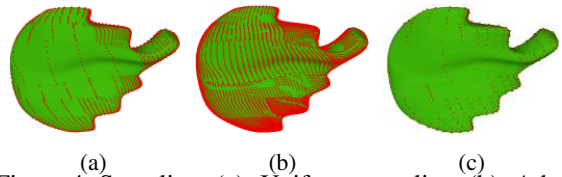


Figure 4: Sampling. (a): Uniform sampling, (b): Adaptive sampling, (c): Chosen sample points

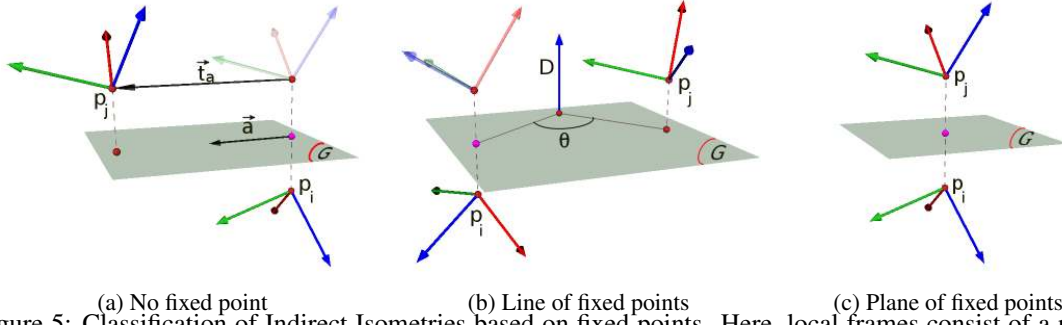


Figure 5: Classification of Indirect Isometries based on fixed points. Here, local frames consist of a normal (red vector) and two principal directions (blue and green vectors).

tation of the frame is necessary, for example to distinguish direct from indirect transformation. The normal vector well oriented (and coherently for the whole surface) by the parameterization, but we modify the direction of tangent frame vectors. For each pair (p_i, p_j) , we identify the orientation of principal vectors at p_j that is the most coherent to direction associated to those at p_i . Suppose that the frame at point p_i is fixed, in other words, the direction of vectors t_1^i and t_2^i is arbitrarily fixed. Consider now the frame at p_j . Each of the tangent vector at p_j can be oriented arbitrarily. Considering both tangent vectors, there are four possible different orientations of principal vectors at p_j .

We project the neighbours of p_i into the tangent plane, and order them into a sequence by turning around p_i . This gives us a reference list of curvatures. The four lists of neighbours of p_j corresponding to the four possible orientations of t_1^j and t_2^j are compared to the reference list. The chosen directions are thus the one that minimizes the sum of squares of differences between its list and the reference list.

Figure 3 shows a case of a plane symmetry where the initial orientation of vectors would have led to identifying a (wrong) direct transformation between points p_i and p_j .

5 ISOMETRY SPACES

Instead of considering all transformations in a 6-dimensional transformation space [Mit13a], we first partition the isometries and map them into one of the five isometry spaces. The advantage of these classifications is two fold: it simplifies the clustering, but also, it expresses the transformation in a space with the appropriate dimension. As an example, clustering translations in the original 6-dimensional transformation space requires the clustering algorithm to discriminate between points that belong to a degenerated 3-dimensional subspace. In our approach, the clustering will be applied directly in this subspace, taking into account only the relevant parameters.

5.1 Computation of the isometry

Given a points pair (p_i, p_j) as in the figure 1, we would like to evaluate the transformation from p_i to p_j so that p_i move to p_j 's position and that the computed orthonormal frame at p_i aligns to the frame at p_j . We denote R_{ij} the rotation between these two frames and t_{ij} the corresponding translation. The computation is as follow:

$$R_{ij} = \begin{bmatrix} n^i \\ t_1^i \\ t_2^i \end{bmatrix}^T * \begin{bmatrix} n^i \cdot n^j & n^i \cdot t_1^j & n^i \cdot t_2^j \\ t_1^i \cdot n^j & t_1^i \cdot t_1^j & t_1^i \cdot t_2^j \\ t_2^i \cdot n^j & t_2^i \cdot t_1^j & t_2^i \cdot t_2^j \end{bmatrix} * \begin{bmatrix} n^j \\ t_1^j \\ t_2^j \end{bmatrix} \quad (7)$$

$$t_{ij} = p_j - R_{ij} * p_i \quad (8)$$

The transformation R_{ij} is an orthogonal matrix, i.e. $R_{ij} \in O(3)$, thus $T_{ij} : p_i(n^i, t_1^i, t_2^i) \mapsto p_j(n^j, t_1^j, t_2^j)$ is then an isometry. Hence, T_{ij} belongs to $Is(X)$, the isometry group. We denote \vec{T}_{ij} the associated linear transform, that is, the transform of matrix R_{ij} .

5.2 Classification of isometries

Affine isometry in three dimensional space, can be classified by considering the nature of its fixed points, according to the following theorem [Tis88a].

Theorem 1 Given $T \in Is(X)$, there exists a unique couple $(g, t_{\vec{d}})$ where g is an isometry having a non empty set of fixed points G and here $t_{\vec{d}}$ is a translation of $\vec{d} \in \vec{G}$ such that $T = t_{\vec{d}} \circ g$. Additionally:

- $T = g \circ t_{\vec{d}}$ and $\vec{G} = E(1, \vec{T})$, the vector subspace associated with the eigenvalue 1.
- $T = g$ and $\vec{d} = 0$ if and only if T has at least one fixed point.
- If T has no fixed point, $\dim G \geq 1$.

In our case, suppose that \vec{T} is not the identity and $\alpha = \dim E(1, \vec{T})$, \vec{T} is direct if $\det(\vec{T}) = 1$ and \vec{T} is indirect if $\det(\vec{T}) = -1$. We can deduce the isometry type of T depending on its fixed points, as follow:

Direct Isometry

1. A line (D) of fixed points ($\alpha = 1$, $\vec{d} = 0$): T is a rotation around the line (D) directed by $\vec{n} \in E(1, \vec{T})$.

2. *No fixed point* ($\vec{d} \neq 0$): T is either a translation of \vec{d} or the composition of a rotation around (D) directed by \vec{d} and a non-zero translation colinear to (D) .

Indirect Isometry

1. *A unique fixed point* A ($\alpha = 0$, $\vec{d} = 0$): T consists of a rotation around an axis (D) directed by $\vec{n} \in E(-1, \vec{T})$ and passing through A , and a reflection relative to the plane (G) containing A and perpendicular to (D) (figure 5a).
2. *A plane* G of *fixed points* ($\alpha = 2$, $\vec{d} = 0$): T is a symmetry relative to the plane G that is defined by $\vec{n}_{1,2} \in E(1, \vec{T})$ (figure 5b).
3. *No fixed point* ($\vec{d} \neq 0$): T is composed of a symmetry relative to a plane G whose the normal $\vec{n} \in E(-1, \vec{T})$, and a non-zero translation parallel to this plane (figure 5c).

Table 1 details the classification of isometries into five subsets. These groups will be treated separately to detect similar patches either among these surfaces or in a surface itself.

While the groups of direct isometries identify approximated patches by rotating and/or translating, the group of indirect isometries determine approximated ones by reflecting.

Iso \ FP	Indirect	Direct
Without	$T_{ij} = s_G \circ t_{\vec{d}}$	$T_{ij} = r(D, \theta) \circ t_{\vec{d}}$
Line of	$T_{ij} = s_G \circ r(D, \theta)$	$T_{ij} = r(D, \theta)$
Plane of	$T_{ij} = s_G$	Not possible

Table 1: Classification of the isometries based on isometry types (Iso) and nature of fixed points (FP); s_G is a symmetry relative to the plane G ; $t_{\vec{d}}$ is a translation of vector \vec{d} ; $r(D, \theta)$ is a rotation of angle θ around axis (D) .

5.3 Comparison of two isometries

We now have five different transformation spaces, and for each, will apply a clustering algorithm. The clustering need to have a distance in each of these spaces, that is, we derive distances for two isometries of the same type.

For direct isometries, the components of isometries are the rotation axis (D) and angle θ , and the translation $t_{\vec{d}}$. As the rotation axis and the translation have the same direction, the translation vector \vec{d} and a point P on the axis are sufficient. For comparing the rotations we use the angles and the distance between the two axes, and the difference of the angles; for translations, we still compare the length of the translation vectors (the angle is the same as for the axes).

For indirect isometries, the analysis is identical to the direct setting, except for the symmetry plane G . The

comparison between planes consists in comparing the normals to these planes and computing the distance between the mid-point and the plane.

In the following, we denote $d(T, T')$ the distance between the two isometries T and T' corresponding to the two point pairs (p_i, p_j) and $(p_{i'}, p_{j'})$; M_{ij} , $M_{i'j'}$ the midpoints of $[p_i, p_j]$ and $[p_{i'}, p_{j'}]$; $dist(P, G)$ denotes the distance from a point, line or a plane to another one.

Direct isometries

$$d(T, T') = (1 - |\cos(D \cdot D')|) + \frac{|\langle \theta - \theta' \rangle|}{2\pi} + \omega_1 dist(D, D') + \omega_2 (||t|| - ||t'||)| \quad (9)$$

Indirect isometries

$$d(T, T') = (1 - |\cos(\vec{n} \cdot \vec{n}')|) + \omega_1 (dist(M_{ij}, G') + dist(M_{i'j'}, G)) + \frac{|\langle \theta_{ij} - \theta_{i'j'} \rangle|}{\pi} \quad (10)$$

The weight ω_i are chosen as the diagonal of the bounding box of the model and so that the terms all vary between 0 and 1.

6 CLUSTERING

After computing the isometries as described in the previous section (Section 5), the clustering step aims at grouping pairs of points having approximatively the same isometry. This step is based on a spectral approach called spectral clustering and differs from the Mean Shift algorithm [Mit13a] which requires difficult parameters tuning.

6.1 Method

Introduced in machine learning by Shi et al. [Shi00a, Von07a], the spectral clustering is an unsupervised method that consists in extracting dominant eigenvectors of a *normalized Gaussian affinity matrix*. These eigenvectors span a low dimensional spectral embedding in which projected data are grouped into clusters. We describe the different steps of this clustering method below.

Let $d(T, T')$ be the distance between the two isometries T and T' in the same class corresponding to the two point pairs (p_i, p_j) and $(p_{i'}, p_{j'})$. Note that $d(T, T')$, and consequently the affinity measure (11), will change depending on the class of the isometries (as described in section 5.2). Let $S = \{(p_i, p_j)\}_{\{i,j=1..N_l\}} \in \Gamma_l$, $l \in [1, 5]$ be the set of N_l pairs in the l -th isometry space and let k be the number of clusters.

This method is based on Gaussian affinity measure, its parameter and their spectral elements. It uses inherent properties of the Mercer kernel (here, the Gaussian kernel) that implicitly projects data in a large scale dimension space where data will be linearly separable. In other words, the Gaussian measure defines a nearness criterion in the linear vector space and weights the

SpectralClustering (S, k)

Construct the affinity matrix $A \in \mathbb{R}^{N_l \times N_l}$ defined by:

$$A_{ii'} = \begin{cases} e^{-\frac{d(T, T')^2}{(\sigma/2)^2}} & \text{if } (p_i, p_j) \neq (p_{i'}, p_{j'}), \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

Construct the normalized matrix : $L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ with $D_{i,i} = \sum_{r=1}^{N_l} A_{ir}, \forall i \in \{1, \dots, N_l\}$.

Construct the matrix $X = [X_1 X_2 \dots X_k] \in \mathbb{R}^{N_l \times k}$ by stacking the k largest eigenvectors of L .

Construct the matrix Y by normalizing rows from matrix X .

Consider each row of matrix Y as a point in \mathbb{R}^k and group them into k clusters with *K-means* method.

Assign the original point pair (p_i, p_j) to class θ if and only if the i^{th} row of matrix Y is assigned to class θ .

Algorithm 1: Spectral Clustering

matching scores. Moreover, classes of arbitrary shapes (in particular, non convex) may be defined [Von07a]. Furthermore, this algorithm only depends on two parameters which are the Gaussian Affinity parameter and the number of classes k . To make this method fully unsupervised, we adopt a heuristic to determine each parameter [Mou11a].

6.2 Affinity parameter

The expression of the Gaussian affinity, defined in equation (11), depends on the parameter σ . The parameter σ defines a threshold on transformation distances between point pairs (p_i, p_j) . To set it, we consider a uniform distribution of the points, that is, such that all points are equidistant from each other.

Elements of S which defines an isotropic distribution are included in a bounding box of size $D_{max} = \max_{(p_i, p_j) \neq (p_{i'}, p_{j'})} d(T, T')$ in each of the m dimension – $d(T, T')$ is defined in section 5.3. By dividing this box into N_l identical volumes, the (uniform) distance between two points is, noted D_{unif} , is:

$$D_{unif} = \frac{\max_{(p_i, p_j) \neq (p_{i'}, p_{j'})} d(T, T')}{N_l^{1/m}}. \quad (12)$$

where m is the dimension of the isometry space (varies depending on the nature of the isometry). We can consider that if a cluster exists, there are points that are separated by a distance lower than D_{unif} . Similarly, the Gaussian parameter σ is used as a fraction of distance D_{unif} : $\sigma = D_{unif}/2$. Thus this heuristic integrates a notion of density of points in a m -dimensional space, and derives a threshold from which points are considered closed.

6.3 Number of clusters

The choice of number of clusters is a general problem for all unsupervised clustering algorithms [Von07a]. To

determine this number of clusters k , we adopt a try-and-test approach by exploiting the Gaussian affinity matrix A and defining a quality measure based on the ratio of Frobenius norms. Let α_k be a bound on the number of clusters to identify. For a value $k' \in [2, \alpha_k]$, the affinity matrix is indexed per cluster. A block matrix is then defined: off-diagonal blocks represent the affinity between clusters and diagonal blocks represent the affinity within the cluster. From this block structure, we can evaluate a mean ratio, called $r_{k'}$, between all off-diagonal blocs and the diagonal blocks in Frobenius norm. From this, among the values $k' \in [2, \alpha_k]$, the minimum of the ratio $r_{k'}$ defines the optimal number of classes k :

$$k = \arg \min_{k' \in [2, \alpha_k]} r_{k'}. \quad (13)$$

This minimum corresponds to a case where the affinity between clusters is the lowest and the affinity within cluster is the highest. More details in this interpretation can be found in [Mou11a].

7 VALIDATION

Ideally, every class obtained by the clustering is a set of point pairs which belong to a couple of surface patches similar up to an approximated isometry. However, spatial coherence between point pairs is lost during the isometry clustering. Therefore, the purpose of the *validation* is to overcome this problem in order to identify similar patches. We present the validation step within a NURBS patch (section 7.1) and then consider region growing over a B-Rep model, which may include multiple NURBS patches (section 7.2).

7.1 Validation within a NURBS patch

The validation is performed by a region expanding process. Given C_k , a class of points pairs in an isometry space, a pair (p_i, p_j) is selected randomly. The chosen isometry T_{ij} is applied to the eight neighbours of p_i , their images are thus compared to eight neighbours of p_j . If the deviation of any neighbour is under a chosen threshold, the points pair is accepted as belonging to the two similar patches. This process continues iteratively; we further test the neighbours of p_i . It is repeated until all points on the surface are visited, or the condition does not hold, or until all pairs in class are considered. This step generates a candidate for two similar patches. Nevertheless, this process stops at the boundary of the surface. But a 3D object modelled by NURBS based B-rep is composed by several NURBS surfaces.

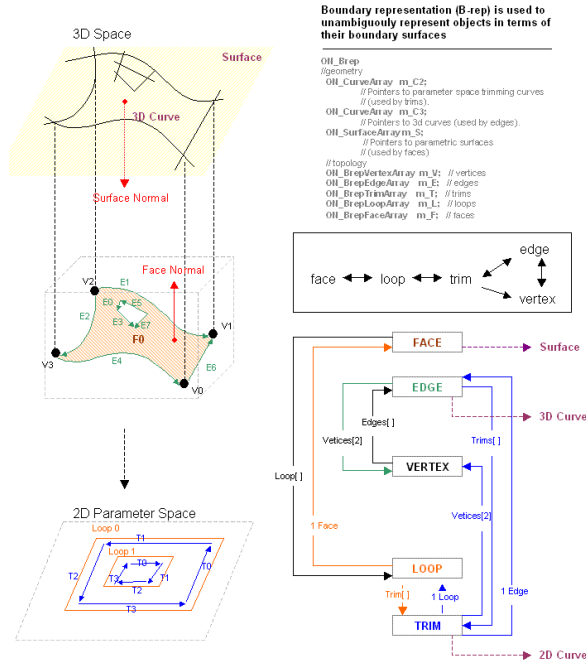
7.2 Validation within a B-rep

The figure 6 represents an overview of the B-rep specification in the context of OpenNURBS. In fact, a NURBS based B-rep is a set of trimmed NURBS that consists in a surface and some trimming contours. The trimming

Find the closest edge e to p
if e is shared with other face **then**
 Determine the adjacent face S
 Take the set P of points on all edges of S
 Find q the closed point to p in P
 Find curvilinear parameters of q

Algorithm 2: Identification of adjacent point

contours define which parts of the surface are kept or removed. In OpenNURBS context, the *loop* is an abstraction of a trimming contour. It is defined by a set of closed trimming curves that are in turn expressed by *trims*. Each *trim* is attached to a 2D curve and an *edge*. The 2D curve defines the curvilinear coordinates of the *trim* within the surface. The *edge* is a 3D curve on the surface and is a boundary. Furthermore, an edge can be shared among multiple trims. Given p the point on

Figure 6: Boundary representation (B-rep) in the context of OpenNURBS (from <http://wiki.mcneel.com/>).

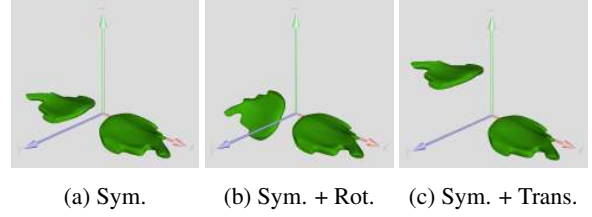
the boundary of the surface where the validation cannot continue. The proposed algorithm 2 identifies a point q on an adjacent surface close to p .

8 EXPERIMENTS

We have implemented the pipeline described in section 3 to identify the similar patches within the following B-Rep models. We use CAD models under OpenNURBS specifications (<http://www.opennurbs.org/>) for our experiments. In general, the main tool that affects directly on the robustness of our pipeline is the *surface orientation* algorithm (section 4.4) and the classification of isometries (section 5.2). In the following, we propose

some test scenarios to validate these tools following by the results on some CAD models of our pipeline.

Since the *surface orientation* algorithm is only applicable for indirect isometries, the models for our test cases exhibit only these types. We proposed three B-rep models of leaves as showed in the figure 7. Given N_{Exp} the number of *expected re-oriented pairs* and N_{Total} the *total number of pairs* computed in each model. Then, the tolerance rate R_{Tol} is the ratio between these two factors.

Figure 7: Proposed CAD models representing the symmetry (Sym.), the rotation (Rot.) and the translation (Trans.) for the *surface orientation* algorithm test cases.

According to table 2, our test cases shows that this algorithm has a tolerance rate up to 80%. Despite the orientation still failed at points whose the opposite neighbors (symmetric via these points in the parameters net) are similar, this algorithm guarantees that the *classification of isometries* is reliable and thus the *similarity detection* is robust. Next, by applying our algorithm of

Model	N_{Exp}	N_{Total}	R_{Tol}
7a	520	621	0.84
7b	509	618	0.82
7c	505	621	0.81

Table 2: Tolerance rate of the *surface orientation* algorithm.

Automatic Spectral clustering [Mou11a], the results of clustering in the figure 8 illustrate the effectiveness between *Euclidean transformation* approach [Mit13a] and our approach of *classification of Isometry*. This figure represents three leaves in a model that have two symmetric pairs of leaves. Besides, every line that connects two points having the same signature corresponds to a point in transformation space. Additionally, lines with the same color are in the same cluster (i.e. the same transformation in general). In this test case, we use the computation of Euclidean transformation and the distance metric as represented by Mitra et al. [Mit13a]. We can observe that while there are some wrong classified points in the *Euclidean transformation* approach (figure 8a), our approach can address this problem (8b). In other words, with the aid of the classification of isometries, the output of the clustering algorithm was significantly improved. Moreover, the use of *Automatic Spectral clustering* algorithm also contributes to the robustness of our pipeline. In fact, the results shown in this figure are obtained without tuning any parameter.

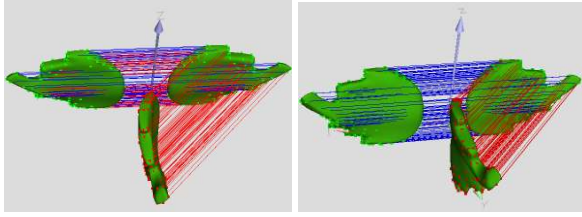
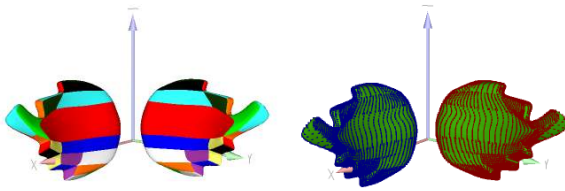


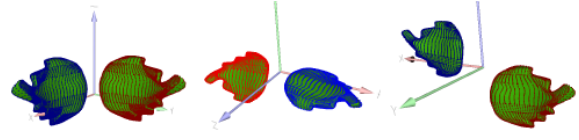
Figure 8: Comparison of the effectiveness between the *Euclidean transformation* approach 8a and the *classification of Isometry* approach 8b.

Also, the figure 9 presents the result of our proposed validation within B-rep. The figure 9a shows that there are two separated B-reps that are formed by several trimmed NURBS surfaces displayed by different colors. As in the figure 9b, the validation has successfully validated all the points over the surface of these B-rep objects.

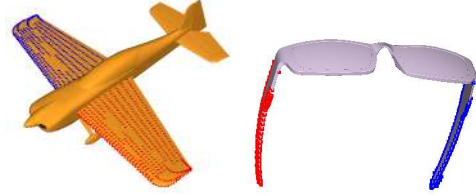


(a) Original B-reps (b) Result
Figure 9: Result of validation within a B-rep.

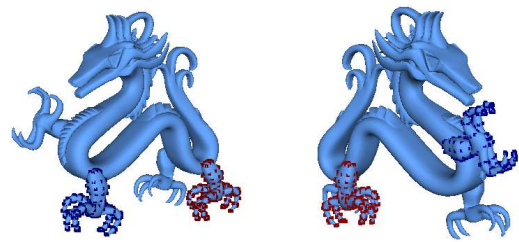
Finally, the figures 10, 11, 12 and 13 represent the final results of our experiments on some CAD models downloaded from GrabCAD (<http://grabcad.com/>). These results represent different isometries detected by our proposed pipeline. The first set of leave models exhibit the indirect isometries. In fact, while the figure 10a shows a symmetry, the figure 10b represents a symmetry following by a rotation axis, and a symmetry following by a translation is detected in the figure 10c. Also, the figures 12a and 12b describe the direct isometries between the four legs of a dragon: this isometry is decomposed into a translation and a rotation. The figures of the plane and the sunglasses demonstrate the symmetry between different parts in these models. In addition, the figure 11a also demonstrates a direct isometry composed by a rotation axis between the two parts of the plane tail. Next, the figure 13a and 13b describe the similarity detection result of a series of human head in a model, in which, from left to right, every head presents a refinement step on the surface. In other words, there are some deformations between these heads. When applying our pipeline, one of the identified transformations is the translation between the green dots and the blue dots (figure 13a), another is the symmetry inside a B-rep (figure 13b). This result demonstrates that our pipeline works well even if there is a slight deformation between the similar surfaces.



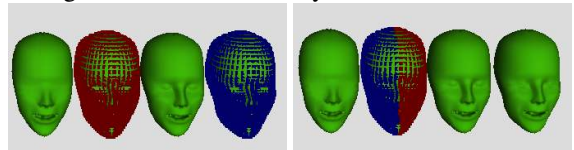
(a) (b) (c)
Figure 10: Similarities in leaves models.



(a) (b)
Figure 11: Symmetry detected in models.



(a) (b)
Figure 12: Direct isometry detected in models.



(a) (b)
Figure 13: Similarities detected in a model of human heads.

9 CONCLUSION

In this article, we propose an algorithm to identify similar parts within objects modelled by NURBS based B-Reps, by adapting and improving the *votes transformation space* approach described by Mitra et al. [Mit13a]. Beside adapting the approach for parametric representations, we have proposed a local coherent frames orientation simply based on the points neighbours. A (robust) globally coherent orientation is then insured at the validation step. The local orientation allows to sort *direct* and *indirect isometries*. Furthermore, based on the analysis of fixed points, local isometries are further partitioned into five subsets. The experiments show that this classification before the clustering steps significantly improves the results. Furthermore, the clustering was further improved by using a fully unsupervised spectral method, for which, unlike for the *Mean-shift* algorithm, parameter tuning is not necessary. In particular, the number of isometries (clusters) to be identified does not need to be known in advance. Finally, the validation step extends the identified isometries among different NURBS patch within the B-rep. We are now able

to recover isometric patches of B-splines or NURBS surfaces or similar to an isometry, or an approximate isometry (like shown in the experiment section). For future work, first we would like to filter the similarity detection by filtering similarities between control points. Second, we would like to exploit the isometries for applications: by linking the control structures corresponding to these patches, to offer the possibility to coherently edit or segment the models. Moreover, we could use the similarity to limit the storage size of the model.

10 REFERENCES

- [Baj97a] Bajaj, C., Lee, H. Y., Merkert, R., and Pascucci, V. NURBS based B-rep models for macromolecules and their properties. *Proc. of Symposium on Solid modeling and applications*, p. 217-228, 1997.
- [Ber08a] Berner, A., Bokeloh, M., Wand, M., Schilling, A., and Seidel, H. P. A graph-based approach to symmetry detection. *Proc. of conf. on Point-Based Graphics*, p. 1-8, 2008.
- [Bok09a] Bokeloh, M., Berner, A., Wand, M., Seidel, H. P., and Schilling, A. Symmetry detection using feature lines. *Computer Graphics Forum*, p. 697-706, 2009.
- [Car06a] Cardone, A., Gupta, S.K., Deshmukh, A., Karnik, M. Machining feature-based similarity assessment algorithms for prismatic machined parts. *Computer-Aided Design (C A D)* 38, p. 954-972, 2006.
- [Cau03a] Caumon, G., Sword Jr, C. H., and Mallet, J. L. Constrained modifications of non-manifold b-reps. *Proc. of the symposium on Solid modeling and applications*, p. 310-315, 2003.
- [Cha08a] Chaouch, M., and Verroust-Blondet, A. A novel method for alignment of 3D models. *Shape Modeling International*, p. 187-195, 2008.
- [Che12a] Chen, X., Gao, S., Guo, S., and Bai, J. A flexible assembly retrieval approach for model reuse. *Computer-Aided Design* 44, p. 554-574, 2012.
- [Chu06a] Chu, C.H., and Hsu, Y.C. Similarity assessment of 3D mechanical components for design reuse. *Robotics and Computer-Integrated Manufacturing* 22, p. 332-341, 2006.
- [Cui11a] Cuilliere, J.C., François, V., Souaissa, K., Benamara, A., and BelHadjSalah, H. Automatic comparison and remeshing applied to CAD model modification. *Computer-Aided Design* 43, p. 1545-1560, 2011.
- [Cyb97a] Cybenko, G., Bhasin, A., and Cohen, K.D. Pattern recognition of 3D CAD objects: Towards an electronic yellow pages of mechanical parts. *International Journal of Smart Engineering System Design*, no 1, p. 1-13, 1997.
- [Dim99a] Dimas, E., and Briassoulis, D. 3D geometric modelling based on NURBS: a review. *Advances in Engineering Software*, no 9, p. 741-751, 1999.
- [Far92a] Farin, G. *Courbes et surfaces pour la CGAO - Conception Géométrique Assistée par Ordinateur*. Masson, 1992.
- [Hil01a] Hilaga, M., Shinagawa, Y., Kohmura, T. Topology matching for fully automatic similarity estimation of 3D shapes. *Proc. on Computer graphics and interactive techniques*, p. 203-212, 2001.
- [Iye05a] Iyer, N., Jayanti, S., Lou, K., Kalyanaraman, Y., and Ramani, K. Three-dimensional shape searching: state-of-the-art review and future trends. *Computer-Aided Design* 37, p. 509-530, 2005.
- [Li11a] Li, X., Yin, Z., Wei, L., Wan, S., Yu, W., and Li, M. Symmetry and template guided completion of damaged skulls. *Computers and Graphics* 35, p. 885-893, 2011.
- [Lip09a] Lipman, Y., and Funkhouser, T. Möbius voting for surface correspondence. *ACM Transactions on Graphics (TOG)*. ACM, p. 72, 2009.
- [Liu13a] Liu, Y.J., Luo, X., Joneja, A., Ma, C. X., Fu, X. L., and Song, D. User-Adaptive Sketch-Based 3D CAD Model Retrieval, 2013.
- [Kaz04a] Kazhdan, M., Chazelle, B., Dobkin, D., Funkhouser, T., and Rusinkiewicz, S. A reflective symmetry descriptor for 3D models. *Algorithmica* 38, p. 201-225, 2004.
- [Ma10a] Ma, L., Huang, A., and Wang, Y. Automatic discovery of common design structures in CAD models. *Computers and Graphics* 34, p. 545-555, 2010.
- [Mit07a] Mitra, N.J., Guibas, L.J., and Pauly, M. Symmetrization. *ACM Transactions on Graphics (TOG)* 26, p. 63, 2007.
- [Mit13a] Mitra, N. J., Pauly, M., Wand, M., and Ceylan, D. Symmetry in 3d geometry: Extraction and applications. *Computer Graphics Forum*, 2013.
- [Mou11a] Mouysset, S., Noailles, J., Ruiz, D., and Guivarch, R. On a strategy for spectral clustering with parallel computation. *Proc. of VECPAR*. p. 408-420, 2011.
- [Shi00a] Shi, J., and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, p. 888-905, 2000.
- [Str61a] Struik, D. J. *Lectures on classical differential geometry*. Courier Dover Publications, 1961.
- [Sun97a] Sun, C., and Sherrah, J. 3D symmetry detection using the extended Gaussian image. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19, p. 164-168, 1997.
- [Tis88a] Tisseron, C. *Géométries affine, projective et euclidienne*. Hermann, 1988.
- [Pod06a] Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., and Funkhouser, T. A planar-reflective symmetry transform for 3D shapes. *ACM Trans. on Graphics*, p. 549-559, 2006.
- [Von07a] Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing* 17, no 4, p. 395-416, 2007.
- [Zab95a] Zabrodsky, H., Peleg, S., and Avnir, D. Symmetry as a continuous feature. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17, p. 1154-1166, 1995.

Generation of Parameterized Models for Vessels Design

Ruben G. Diaz.², Marcelo Dreux¹, Luiz Cristovão G. Coelho²

¹Department of Mechanical Engineering

²Tecgraf-Computer Graphics Group

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225, Gávea

Phone: +55 (21) 3527-1162/1639/1164

CEP 22453-900 - Rio de Janeiro - RJ - BRAZIL

rgomez@tecgraf.puc-rio.br, dreux@puc-rio.br, lula@tecgraf.puc-rio.br

ABSTRACT

This work proposes an integrated environment for modeling, and static and dynamic analysis of vessels. The main advantage of the proposed environment is that it is possible to automatically obtain variants of a specific model in order to achieve a desired configuration, not only in relation to geometry but also concerning the static stability aspect. This environment uses the Lua programming language and it is possible to define global variables to be used as parameters which retrieve or modify modeling values such as length, width, height, and so on. Any model can be parameterized, as a function of user chosen variables, which allows an automatic modeling with the variation of those parameters.

Keywords: Geometric Modeling, Computer Aided Geometric Design, Vessel Stability, Computational Geometry, Mesh Generation and Simplification, Lua.

1 INTRODUCTION

Many different engineering areas deal with complex geometry shapes such as compressors, turbines, vessel units, car, etc and relies on different kinds of modeling systems ([3], [19], [4], etc) that are capable to reproduce these shapes. Those systems can also define a finite element mesh in order to perform numeric simulations.

The area of Computer Aided Ship Hull Design (CASHD) is concerned with modeling systems that can generate different shapes and geometries of a vessel unit in an iterative or automatic manner.

The design and construction of an offshore vessel unit is not a simple task. Many aspects have to be considered before building it, as in all engineering projects. The most used methodology to develop an offshore unit was proposed by Evans [11].

The designer of a vessel unit searches for a solution that best fits the operational requirements, while attending the economic and engineering constraints. During this search the designer will come up with several solutions that can be either unfeasible or do not fulfill the design criteria. Therefore, it is necessary a rational approach to guide the designer in this pursuit.

Bearing this scenario in mind, PETROBRAS (the Brazilian oil Company) has developed two

softwares in collaboration with two research groups: CENPES (*Research Center of PETROBRAS*) and Tecgraf (*Computer Graphic Technology Group*) from PUC-Rio. These softwares are called MG (*Mesh Generator*) [8] and Sstab [7].

MG is used during the modeling phase in order to create the basic geometry of a vessel unit. The meshes for static and dynamic analysis are also generated and then in the phase of static stability analysis the Sstab software is used. The static stability evaluates the capacity of the vessel unit to restore its initial equilibrium after any perturbation. Finally, the Wamit [21] software is used to perform dynamic analysis in order to study the influence of the sea waves against the vessel unit structure.

2 RELATED WORK

Coelho [8] presented a modeling system based on patches (*MG-Mesh Generator*), which makes use of a boundary representation model (*BRep*). Kassab [16] implemented an efficient solution to solve the surface mesh generation problem but representing the regions where the surface curvature changes using a reduced set of non uniform faces. This approach uses a modification of the space quadtree subdivision.

Regular meshes using NURBS surfaces are generated for simulation purposes and in some cases a technique that uses advancing front mesh generation with quadrees is used where it is not possible to create a regular mesh [18]. Another work that uses NURBS surfaces in order to perform ship hull modeling is presented by Rasmussen [20], where the ship modeling is divided in several parts. Each part, like bow, stern, parallel middle body, etc is modeled by a NURBS

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

surface and then these surfaces are merged into a global NURBS surface that represents the whole ship model.

An automatic parametric modeling approach was implemented by Oliveira [10] using the MG software in order to create the geometric model, but at the same time it is necessary to use another software to define the shape of the model. Therefore, it is possible to perform automatic modeling in order to generate many instances of the model, varying some form parameters. After the modeling phase these instances could be used to perform static and dynamic stability analysis.

Mendonça [9] allows the designer to perform a shape optimization of an offshore vessel unit by combining neural networks and a genetic algorithm that makes use of scripts written in Lua language [17].

Moreover, integrating the tools for modeling, analysis, simulation and assessment leads to faster processes and better products. So, Abt [2] proposed a tool integration system for simulation-driven design using XML(Extensible Markup Language) and COM(Component Object Model) interfaces into the FRIENDSHIP CAD environment for ship models. Abt [1] also proposed a new integration of FRIENDSHIP combining CAD and CFD, allowing rapid hydrodynamic evaluation in the ship design.

Birk [6] presented an hydrodynamic analysis optimization which uses a command language approach that interconnects the modeling phase with the hydrodynamic analysis. Harries et al [13] developed an integrated approach that simultaneously covers all relevant aspects of early ship design: main dimensions, hull form, hydrodynamics and safety including intact and damage stability; etc.

Other work for hull optimization is presented by Karri [15], where a simple interpolation scheme is used to generate panels for any conventional ship hull and the main objective is to design a ship hull that matches its operational conditions.

There is a strong tendency in searching an automatization process in the geometry phase modeling and also in the stability and dynamic phase analysis.

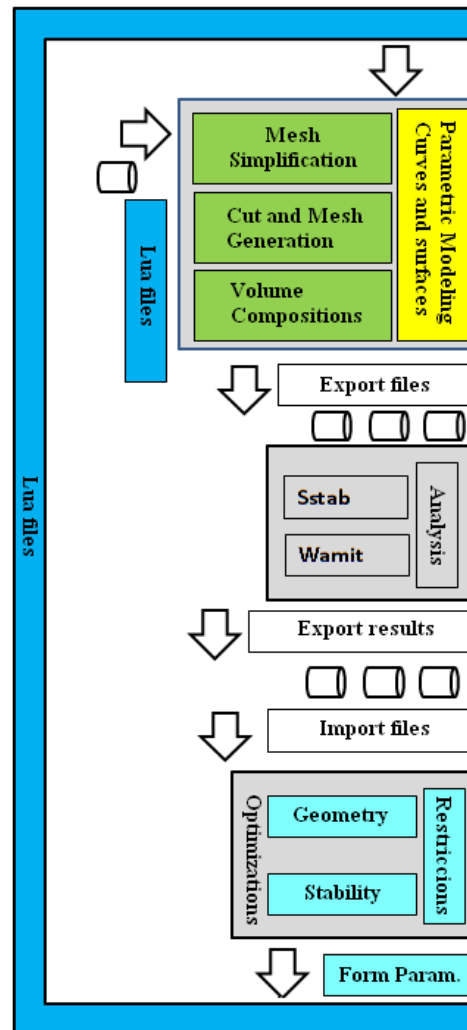
3 MODELING METHODOLOGY

The previous section presented some works that are concerned with automatic geometric modeling but also require an external software to perform this task.

The framework shown in Figure 1 provides to the project designer the necessary tools to create a set of instances of a model, by simply varying some chosen form parameters.

Lua language has been embedded into the MG and Sstab software in order to provide a single environment with the three softwares.

Figure 1: Modeling framework for vessel design



This work presents a methodology to create engineering vessel models using the MG (Mesh Generator) [8] and Sstab [7] software combined with automatic parametric geometry modeling. Scripts written in Lua [17] allow the project designer to obtain a set of different vessel units, according to constraints in the geometry and aspects of the static stability.

All the user actions (the creation or modification of vertices, curves, surfaces, volumes, etc.) during the geometric modeling phase are stored in a script history file and can be reproduced in an intuitive way. These scripts can also be parameterized in order to obtain different variations of the original model.

Geometric constraints are checked as well as static (Sstab) and dynamic (Wamit) stabilities. Once the static and dynamic analysis are performed the results can be exported to an optimization module. If the model does not comply with the geometric or stability restrictions some form parameters can be changed. These changes are achieved by Lua [17] scripts and MG generates a new model. This fact may reduce the process of re-visiting several times some cycles of the spiral methodology design process proposed by Evans [11]. This process goes on until a convergence appears in the simulations.

4 PARAMETRIC MODELING OF A SHIP HULL

One of the softwares used in this work, MG, is a shell modeler based on cross sections curves and it has been widely used in the design of several large offshore oil structures at Petrobras, the Brazilian oil company. This software has an interactive modeling environment based on direct manipulation in 3D space, so it addresses several interesting user-interface issues, that have been discussed elsewhere [12]. Different types of meshes can be generated by the geometric modeler using trilinear, bilinear, planar, BSpline surfaces, etc.

In the next sections some modeling strategies using MG environment will be presented which could be used when designing a realistic engineering model. Some designing stages may be a hard task and some of these strategies presented here could be useful.

4.1 Volume Composition

Two schemes for representing a three-dimensional geometric model into a computer are used: a boundary scheme and a constructive scheme. In the boundary scheme, the geometry of an object is defined by its boundary elements, such as vertices, edges and surface patches, which may be created through an interactive graphics interface. The most common constructive scheme is constructive solid geometry (CSG) [5], in which the final object is obtained by a set of boolean operations applied to a set of primitive objects.

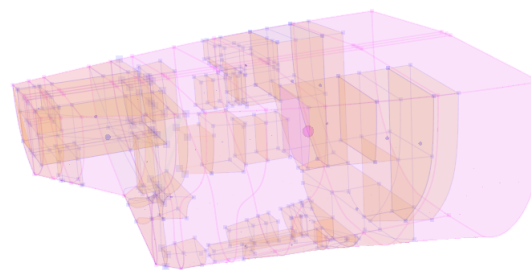


Figure 2: Example of volume composition

Realistic engineering model like vessel structures generally are composed by a hull volume and a set of internal compartments. Defining which mesh is internal or external is particularly important for finite element mesh generation and in this case could be a hard work when there is a lot of adjacent surfaces like it is shown in figure 2 where the stern of a ship has many internal compartments and empty spaces.

In order to define a new volume or an internal compartment that could fit in all empty spaces shown in figure 2 it is necessary to perform addition or subtraction operations, and to make sure that the created volume will be topologically correct. Volume contribution for each face will be positive and negative respectively.

A less complicated model than the one presented in figure 2 is a cylinder with one internal compartment (figure 3a). Sstab software uses the mesh exported by MG for static analysis by computing the volume contribution below a draft plane. Considering the new faces orientation defined (External or Internal) it is possible to see an empty space relative to the smaller compartment defined inside the cylinder where no volume contribution is considered for the final volume computation (figure 3b).

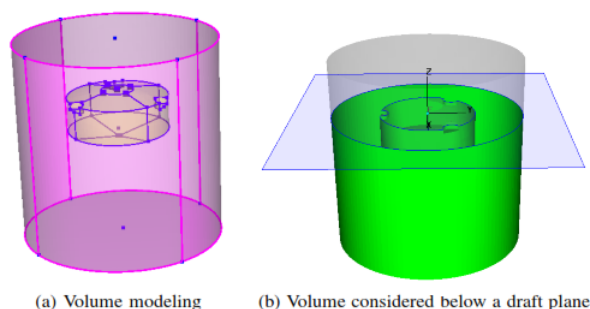


Figure 3: Volume contribution for composed volumes

The new created volume will fit into the empty space left by the other existing volumes. All volumes are modeled individually and then composed into a new one.

This approach is similar to the constructive scheme called CSG where a set of operations is also applied but in this case is more simplified with only two operations, so it was called of Pseudo-CSG because makes easy to create new volume considering a set of existing volumes.

4.2 Virtual Entities

Mirror symmetry is another important feature of an object being modeled from a set of curves and surfaces. Generally a ship hull contains symmetry properties in one or more planes relative to the cross section curves. Models that contain for example 30 surfaces could be modeled by half of the surfaces and curves by just applying mirror transformations.

An important task when defining a ship hull surface is to achieve a faired surface or a set of surfaces that are created using some specific points that belong to the cross section curves of the ship hull (figure 4).

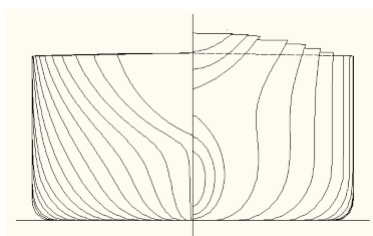


Figure 4: Cross sections of a ship hull.

This section proposes an approach to model an object that has some similar properties using virtual entities. Each virtual entity created is defined as an OpenGL matrix transformation that will be applied to the original surface in order to get the final position of each surface, curve, etc.

Virtual entities may contain any kind of transformation such as: mirror, shear, scale, translation etc. These type of entities could be very useful when designing an engineering model holding a lot of surfaces that have some properties in common. Due this the final model can contain a minimal set of curves and surfaces that represents the whole geometry itself. It is used an OpenGL optimization technique called Display List in order to avoid repeating a set of transformation operations. Display list is just a group of compiled function calls stored for subsequent execution. Once a display list is created it could be called as many times as needed by the final application.

FEM meshes that will be generated considering a model with symmetry properties need to orient its faces according to the mirror operation applied. Figures 5a and 5b show a model containing a set of virtual surfaces and its corresponding created FEM mesh. Meshes containing virtual entities with no symmetry do not need to re-orient their internal faces. This operation is only performed for mirror transformation.

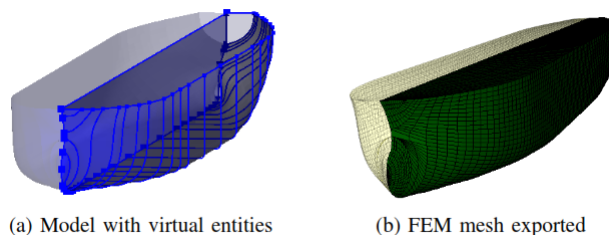


Figure 5: Virtual Entities.

4.3 Volume Sweeps

The sweep volume of a 3D object is a powerful tool that makes possible to transform a surface into a volume. It has been proved to be an excellent aid when planning and designing a 3D model (shell) that contains internal compartments like a 3D vessel structure. Volume sweep is a solid bounded by parametric surfaces undergoing an arbitrary direction. The direction chosen in this work is the normal direction of each parametric surface. Figure 6 shows creation of volumes considering a set of initial selected surfaces and a chosen direction. In the example shown below the chosen direction was along Z axis.

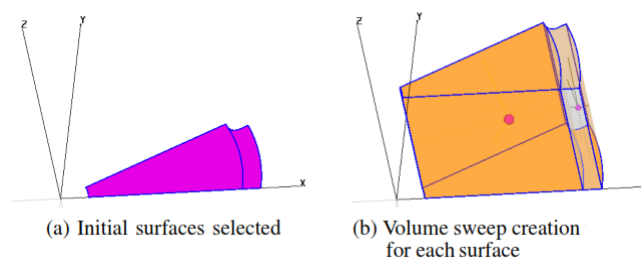


Figure 6: Volume sweep creation.

The building approach shown here leads to model a 3D volume or internal compartments of a vessel structure much more easy as shown in figure 7 where all internal compartments of a cylindrical vessel structure have been modeled using this approach.

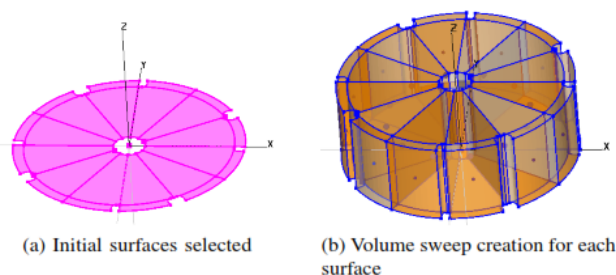


Figure 7: Internal compartments of a cylindrical vessel using sweep volume creation.

Internal compartments in a vessel structure are very important to be modeled in order to simulate a damage situation for example where one or more internal compartments can be flooded. Figure 8 shows the mesh used to perform some simulation according its hydrodynamic or static properties defined by a current situation. Symmetry properties are automatically detected when a mesh is generated to perform simulations. The mesh is generated only in part of the model and then transformed to its symmetrical parts relative to XZ and YZ planes.

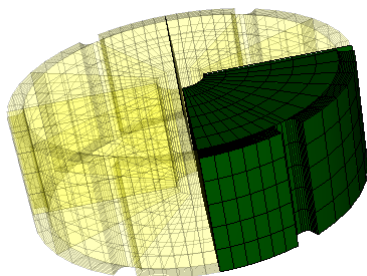


Figure 8: Mesh compartments of a cylinder

4.4 Lua Language Programming

Many different engineering areas deal with complex geometry shapes and the environment used to reproduce this models are adopting extension language or scripts as a way to increase flexibility and modeling facilities.

Parametric modeling allows to describe properties of a model by using geometric descriptors known as form parameters. This descriptors can be used in order to increase a wide range of facilities in the early modeling phase of a CAD model.

It is shown in figure 9 an example of MG capabilities where a ship hull is modelled using different type of surfaces (bilinear, trilinear, Bspline) with their corresponding geometric support and mesh discretization. It is also possible to model sweep surfaces defined by a translational, rotational or generic direction.

The main dimensions of a ship hull, as the one shown in Figure 9, are LBP, Breadth and Depth. LBP specifies the size of the ship hull between perpendiculars relative to X axis. Breadth specifies the size of the ship relative to Y axis and Depth is the height of the ship, relative to Z axis.

It is shown in table 1 values of the parameters LBP, Breadth and Depth that were set to 319, 56 and 30,2 respectively. The value of 165 refers to the middle section of the ship.

Lua scripts have been written that are able to modify the shape of the ship hull but keeping the model always centered in relation to its middle section. Therefore, the Lua function TransformLBP shown in Figure 10,

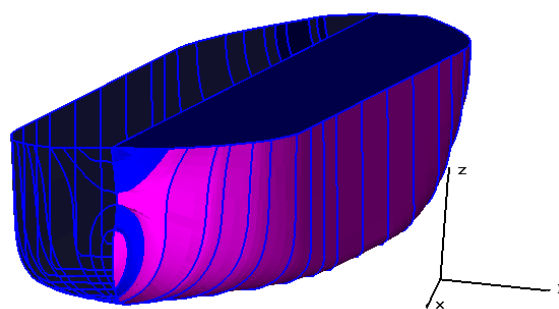


Figure 9: Ship hull modelled by Bspline surfaces, bilinear and trilinear mappings.

Description	Value
Mid Ship with respect to (wrt) Keel	165,00
Length Between Perpendiculars wrt Mid Ship	319,00
Molded Breadth wrt Mid Ship	56,00
Molded Depth wrt Mid Ship	30,20

Table 1: Initial dimensions of the ship hull.

is used to parameterize the ship in relation to the LBP parameter.

```
function TransformLBP(idv1,idv2,NLBP)
1 // get the vertex positions of idv1 and idv2
2 a=mgGetVertex(idv1)
3 b=mgGetVertex(idv2)
4 // compute LBP distance
5 LBP = a.Distance(b)
6 // get factor scale relative to the new value of LBP
7 factor = NLBP/LBP
8 // get the midsection point
9 LBPMx=(a.x+b.x)*0.5
10 LBPMx=(a.y+b.y)*0.5
11 LBPMz=(a.z+b.z)*0.5
12 mgSelect("all")
13 mgTranslateXYZ(LBPMx,LBPMx,LBPMz)
14 mgScaleXYZ(factor,1,1)
15 mgTranslateXYZ(-LBPMx,-LBPMx,-LBPMz)
16 return factor,LBPMx,LBPMx,LBPMz
end
```

Figure 10: Lua script to parameterize the ship with relation to LBP parameter

The function TransformLBP computes a scale factor between the current *LBP* value and the new value *NLBP*, passed as a parameter. The current *LBP* value is computed using the reference points passed to the function by *idv1* and *idv2* parameters. After the scale factor is computed, all entities of the model (vertex, surfaces, volumes, etc) are selected and then the appropriated transformations are applied.

The translation is performed in relation to the middle point of LBP, in order to keep the model centered with its midship section, and then the scale factor is applied. The function to transform the Breadth ship parameter is similar to the TransformLBP but all transformations must be applied relative to Y axis.

The function TransformDepth, shown in Figure 11, applies transformations in relation to the Depth parameter of the ship hull. The same transformations of TransformLBP are also applied in this function, but it is necessary to apply an extra transformation according to the scale factor value.

```
function TransformDepth(idv1,idv2,NDepth)
  1 // get the vertex positions of idv1 and idv2
  2 a=mgGetVertex(idv1)
  3 b=mgGetVertex(idv2)
  4 // compute Depth distance
  5 DEPTH = a.Distance(b)
  6 // get factor scale relative to the new value of NDEPTH
  7 factor=NDEPTH/DEPTH
  8 // get the midsection point
  9 DEPTHMx=(a.x+b.x)*0.5
  10 DEPTHMy=(a.y+b.y)*0.5
  11 DEPTHMz=(a.z+b.z)*0.5
  12 mgSelect("all")
  13 mgTranslateXYZ(DEPTHMx,DEPTHMy,DEPTHMz)
  14 mgScaleXYZ(1,1,factor)
  15 mgTranslateXYZ(-DEPTHMx,-DEPTHMy,-DEPTHMz)
  16 // keep keel ship at original point relative to Z axis
  17 if (factor>1) then
  18   mgTranslateZ((NDEPTH*0.5)-DEPTHMz)
  19 end
  20 if (factor<1) then
  21   mgTranslateZ(-(DEPTHMz-(NDEPTH*0.5)))
  22 return factor,DEPTHMx,DEPTHMy,DEPTHMz
  23 end
fim
```

Figure 11: Lua script to parameterize the ship with relation to Depth parameter

Each parameter LBP, Breadth or Detph of the ship hull can be individually modified or the three modifications can be done at the same time.

Figure 12 shows the modified hull of Figure 9. In case that the hull vessel has internal (hull, ballast, diesel, void spaces, etc) or external compartments (super structure, etc), all of them will be also transformed in order to fit the new model geometry. The transformations of the ship hull internal compartments are important in order to guarantee the consistency of the generated model.

In other words, there are geometric correlations between the internal tanks and the ship hull. This correlation is attended by Lua scripts implemented in this work.

Once the geometric modeler automatically modifies the hull shape, it is exported to Sstab software in order to analyze its static stability. If the model is symmetric

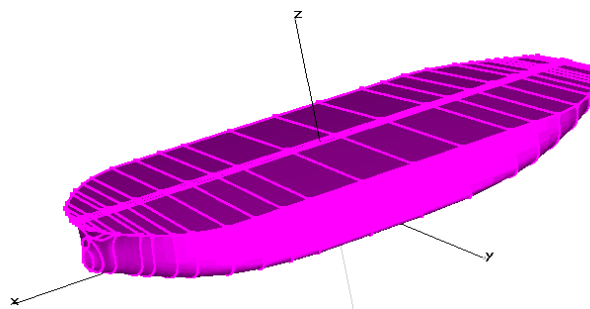


Figure 12: Ship hull parameterized by Lua function TransformShip.

in relation to the diametral plane, then the generated meshes should also be symmetric. In order to avoid the occurrence of trim and heel, due to an inconsistent mesh generation, only half of the model is stored.

5 FORM SHAPE STABILITY ANALYSIS

The behavior of an offshore unit is evaluated in two different situations:

- The static intact stability considering wind actions;
- The static stability in a damaged condition.

These situations are regulated by international organizations such as IMO (*International Maritime Organization*) [14], NMD (*Norwegian Maritime Directorate*), etc

In this work, an offshore ship unit (Figure 9) has been studied to evaluate its behavior when submitted to an overweight with the intention to simulate a damaged condition.

In the design project of a vessel unit, the weight conditions are organized in classes that usually separate the intact conditions from the damaged conditions, according to IMO. In order to test a damaged condition, an intact condition must be simulated beforehand.

The stability requirements after a damaged condition, according to the MARPOL 73/78 rules, are listed below and are shown in Figure 13.

- In the final stage of flooding, the absolute angle equilibrium shall not exceed 25 degrees.
- The stability in the final stage of flooding shall be investigated and may be regarded as sufficient if the righting lever curve has at least a range of 20 degrees beyond the position of equilibrium.
- The maximum residual righting lever shall not exceed 0.1 meters within the 20 degrees range indicated above.
- The area under the curve within this range shall not be less than 0.0175 meters radians.

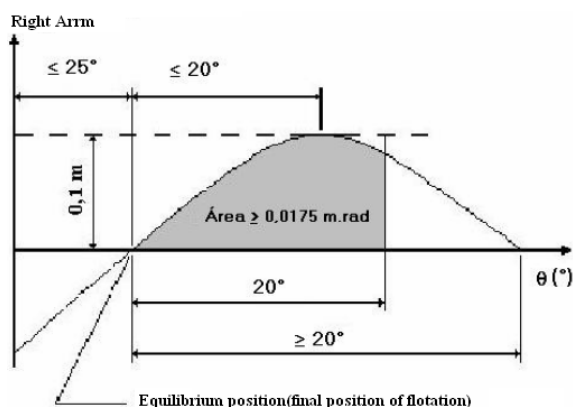


Figure 13: Stability damage rules

Initial Damage Conditions The damaged condition considered in this example is the bulkhead's ship vessel collision that causes the flooding of two stern compartments and the moorings storehouse. Figure 14 shows the same ship presented in Figure 9 but considering all its internal compartments. Damaged tanks are indicated by a black circle.

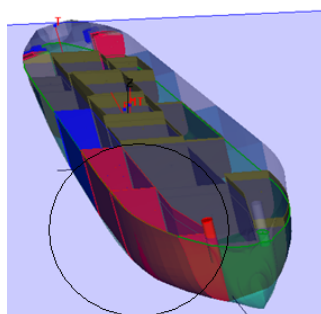


Figure 14: Initial damaged conditions

Table 2 presents the results of the static equilibrium analysis and their corresponding stability diagram for the current damaged condition. The initial dimensions of the ship hull are the same listed in table 1, and the VCG (*Vertical Center of Gravity*) of the light weight within the ship vessel structure is located in 33 meters.

All the restrictions imposed by the MARPOL 73/78 regulations are being accepted but one restriction concerning to the minimal range of stability angles of 20 degrees is currently rejected. This result can be seen in table 2 where the range angle is highlighted in red text.

The next section of this work will present an approach to overcome this damaged situation by testing different variations of the ship model until all MARPOL 73/78 restrictions rules are being accepted.

Automatic Weight Estimation In order the ship meets all the requirements imposed by MARPOL 73/78, it's geometry must be changed by varying the light weight

Criterion	Value	Eval
Equilibrium free-board (lowest flooding point height)	5,554 > 0,000	OK
Angle of deck edge immersion	18,392	Ok
Equilibrium heel angle	14,297 < 25,000	OK
Equilibrium trim angle	0,648	-
Stability range (beyond equil. position with 0.1m level)	18,627 > 20,000	NO
Area under GZ curve in Stability range	0,204 > 0,018	OK
Maximum GZ in Stability range	1,009 > 0,100	OK
Theta WE > Theta 0	27,949 > 14,297	OK

Table 2: Marpol Damaged Condition Stability Table Results

of the vessel structure and the draft equilibrium in order get all MARPOL 73/78 requisites to be approved.

```

function FindLightWeight(factor,toler)
1  // set model into an intact condition
2  sstSetIntactCondition()
3  while true do
4    // get current weight
5    weight=sstGetLightWeight()
6    // compute equilibrium
7    sstComputeEquil()
8    // get total volume and volume displaced
9    totalVol=sstGetHullVol()
10   floodedVol=sstGetHullVolDisplaced()
11   percentVal=floodedVol/totalVol
12   // check user tolerance
13   if (percentVal-factor <= toler) then
14     break
15   end
16   if (percentVal > factor) then
17     weight=weight+(weight*0.01)
18   end
19   if (percentVal < factor) then
20     weight=weight-(weight*0.01)
21   end
22 end
23 end
fim

```

Figure 15: Lua script to find light weight proportional to 63% of the vessel.

A parameter considered to redefine the hull shape is the breadth, but resizing the hull structure does not guaranty a significant gain in the stability range because it is directly linked with the initial GM

(distance between the mass center and the metacenter) computed in its intact equilibrium condition.

Once the ship geometric model have been exported into Sstab software, it was decided to keep constant the relation between submersed volume by total volume in about 63% taking as reference the intact condition. This way, the light weight increases proportional the variation of the submersed volume.

The function *FindLightWeight* (Figure 15) developed in Lua language searches for the light weight that reaches the 63% of flooded volume, according of the input geometric model. This process is done iteratively adding and subtracting weights until a determined tolerance that defines the correct light weight of the vessel is reached, where the total flooded volume must be proportional to 63% equal to the intact condition. At each search step of the *FindLightWeight* function the vessel equilibrium is computed in order to check the flooded volume percentage.

The Lua function *ComputGammaAngles* shown in Figure 16 checks if the current damage condition is approved by the MARPOL 73/78 requirements. Each damage condition computes the current static stability diagram, and the minimal range angle value is evaluated. This value must be greater than 20 degrees.

```
function ComputGammaAngles(DC)
1  // enable damage condition DC
2  sstSetDamageCondition(DC)
3  // use MARPOL criterias
4  sstSetMarpolCriteria()
5  sstComputeEquil()
6  // compute stability diagram
7  sstComputeStabilityDiagram()
8  val=sstGetGammaRange()
9  // check range value
10 if (val > 20) then return 1
11 else return 0 end
12 end
] fim
```

Figure 16: Lua script to compute stability range.

In order to find the ship that meets all the requirements imposed by MARPOL 73/78, considering a set of ship models that have their breadth parameters varying between 56 and 70 meters. The function *CheckModels* (Figure 17) were developed using the functions *FindLightWeight* and *ComputGammaAngles* presented above. This function loads a file that has stored the intact and damage condition for each breadth value and then sets the VCG in 33 meters (function *FindLightWeight*) and then checks the minimal range angles of stability for both conditions (function *ComputGammaAngles*).

```
function CheckModels(VCG,BMin,BMax)
1  currentdir="D:/mg/data/tese/FPSOwith_sst/"
2  // VCG value is set equal to 33
3  // BMin and BMax refer a range
   // of Breadth values
4  for Boca=BMin,BMax,1 do
5    if (sstLoad(currentdir.."FPSO_"..Boca.."mg"))
       then
6      SetVCG(VCG)
7      FindLightWeight()
8      ret=CheckGammaAngles()
9      if (ret) then print("gamma range achieved")
10     else print("gamma range not achieved") end
11   else
12     print("file not found")
13   continue
14   end
15 end
16 end
fim
```

Figure 17: Lua generic script to compute stability range of a set of ship models.

6 RESULTS

Different variations of the model shown in Figure 9 were generated varying the breadth parameter and using Lua functions to perform this process in an automatic way. The geometry variation does not changes the value of the block coefficient that stand in around 0.70 meters. So, it could be say that fluctuation characteristic of the vessel was kept intact.

The static stability properties results with the tested models according to the breadth and weight variation parameters are presented in Table 3.

The model reference or initial condition is highlighted with horizontal lines that forms a box. Therefore it can see that the breadth parameter (B) was modified up and down the breadth reference that is 56 so only three models from Table 3 (bold data) achieved the minimal range stability for the current damage condition.

The partial volume shown in the table is the total flooded volume with relation to the added weight. The column Displacement indicates the total displace weight of the vessel. The data in columns 1 to 7 were extracted with reference to its intact condition, so there is no damage being considered. Data from columns 8 to 10 were extracted considering the damage condition.

Considering the models that were approved by the MARPOL 73/78 restrictions in its damage condition, in case that the vessel needs to supports the current weight loading, the most indicated geometry to be adopted as

B	Draft	Displacement	Total vol	Parcial vol	Vol. %	Weight	KG	GM	Range
50	20,45	272501,4	420870,8	265836	0,63	25000	15,75	5,20	20,26
52	20,48	284098,8	437712,1	277152	0,63	35000	16,01	5,79	19,04
53	20,51	290179,0	446125,0	283084	0,63	41000	16,17	6,07	18,71
54	20,48	295194,3	454540,0	287977	0,63	46000	16,26	6,42	18,80
56	20,52	306998,8	471375,4	299499	0,63	57806	16,56	7,02	18,62
57	20,44	311197,3	479798,9	303591	0,63	62000	16,62	7,44	19,09
58	20,37	315391,4	488215,5	307684	0,63	66194	16,69	7,86	19,54
58,2	20,37	316715,3	489893,6	308976	0,63	67518	16,72	7,92	19,49
58,5	20,35	318701,2	492418,8	310913	0,63	69503	16,78	8,00	22,46
59	20,38	321197,6	496631,4	313349	0,63	72000	16,84	8,19	19,53
60	20,39	327003,8	505044,1	319014	0,63	77806	16,98	8,54	19,52
62	20,42	338616,3	521879,8	330343	0,63	89418	17,30	9,24	19,48
70	20,39	382199,5	589219,1	372867	0,63	133000	18,30	12,6	19,81
72	20,41	393831,4	606054,0	384215	0,63	144632	18,60	13,49	34,10

Table 3: Breadth parameter variations of a ship hull.

reference would be the model whose breadth parameter is 58.5 meters because this model could have the least economical cost to be considered.

The implemented Lua scripts for searching the optimal geometry of the vessel that supports the current weight loading and the damage condition may consider or not all the restrictions imposed by MARPOL 73/78 (minimal stability ranges angles, right arm, etc).

In this work the minimal stability ranges angles were only considered because it was the only condition that was not satisfied for the damage condition simulated considering a hull vessel with breadth equal to 56 meters.

The varying process modifies the breadth parameter up and down considering the reference model (breadth equal to 56) does not mean that the minimal range stability angles (column 10) will have a proportional variation. This fact could be seen in the Table 3 where the breadth variation parameter can get a gain or loss in the minimal range stability angles.

Hence, the current problem is to look up for the local minimum that minimizes a certain objective function that project designer wants to achieve and at the same time all the restrictions requirements by the classifying societies (IMO, DNV, etc) must be approved.

Criterion	Value	Eval
Equilibrium free-board (lowest flooding point height)	7,226 > 0,000	Ok
Angle of deck edge immersion	19,795	Ok
Equilibrium heel angle	14,144 < 25,000	Ok
Equilibrium trim angle	0,520	-
Stability range (beyond equil. position with 0.1m level)	22,456 > 20,000	Ok
Area under GZ curve in Stability range	0,398 > 0,018	Ok
Maximum GZ in Stability range	1,662 > 0,100	Ok
Theta WE > Theta 0	29,636 > 12,144	Ok

Table 4: Approved Marpol Damaged Condition Stability Table Results

Table 4 show the corresponding numeric table resume of all MARPOL 73/78 restrictions that were considered after the equilibrium computation of the vessel and can be appreciated an stability range of 22.456 value.

7 CONCLUSIONS

This work has as principal contribution the building of an embedded scripting language inside a well known softwares that are actually been used at PETROBRAS, the Brazilian oil company for modeling shells of vessel

unit based on cross sections and for static stability analysis. Lua language was chosen because is freely available for both academic and commercial purposes and is a general-purpose embedded programming language designed to support procedural programming with data-description facilities and the most important is that Lua was born at PUC-Rio.

The embedded Lua language let the user to define variables in Lua. This variables can be used as input parameters for parametric modeling functions that sintetize data as: length, hight, etc of any kind model.

A set of Lua languages commands was defined in order to perform automatic modeling in MG. The Lua language was also embedded into the Sstab software in order to be able to write all the Lua functions shown in this section. The Lua language enables that many iterative process to be done in an automatic way.

This command sets sintetize the process of creating any kind of basic geometric entities (vertices, curves, surfaces, etc). A set o Lua language commands was also create for the Sstab software in order to perform automatic analysis as the scripts shown in the section Form Shape Stability Analysis.

8 ACKNOWLEDGEMENT

The authors are grateful to Department of Mechanics of PUC-Rio and Tecgraf-PUC-Rio for the opportunity to develop this work. The first author is also grateful to CNPq, the Brazilian government research council which partially sponsored this research.

REFERENCES

- [1] C. ABT and S. Harries. A new approach to integration of cad and cfd for naval architects. *6th COMPIT*, 2007.
- [2] C. ABT, S. HARRIES, S. WUNDERLICH, and B. ZEITZ. Flexible tool integration for simulation-driven design using xml, generic and com interfaces. *8th COMPIT*, 2009.
- [3] www.spatial.com/products/3d/modeling/acis.html.
- [4] www.ansys.com.
- [5] M. C Arruda. Operações booleanas em sólidos compostos representados por fronteira. Master's thesis, Departamento de Engenharia Civil, PUC-Rio, 2005.
- [6] Lothar Birk. *Hydrodynamic Shape Optimization of Offshore Structures*. PhD thesis, Technische Universit at Berlin, 1998.
- [7] L.C.G Coelho, C.G. Jordani, M.C. Oliveira, and I.Q Masetti. Equilibrium, ballast control and free-surface effect computations using the sstab system. *International Conference of Stability of Ships and Ocean Vehicles -Stab*, 8:377–388, 2003.
- [8] Luiz Cristovão Coelho. *Modelagem de Cascas com Interseções Paramétricas*. PhD thesis, Departamento de Informatica, PUC-Rio, 1998.
- [9] Carlos Eduardo Luz Riodades de Mendonça. *Um Sistema Computacional para Otimização Através de Algoritmos Genéticos e Redes Neurais*. PhD thesis, COPPE-UFRJ, 2004.
- [10] Mauro Costa de Oliveira. Offshore platforms sizing optimization through genetic algorithms. In *20th Deep Offshore Technology International Conference(DOT 2008)*, 2008.
- [11] J H Evans. Basic design concepts. Technical report, 1959.
- [12] L.C. Gomes Coelho and C.S. de Souza. Comunicação de problemas e soluções geométricas em uma interface 3d. In *Anais do VII SIBGRAPI*, pages 233–240, 1995.
- [13] S. HARRIES, F. Tillig, M. Wilken, and G. Zaraphonitis. An integrated approach for simulation in the early ship design of a tanker. *10th COMPIT*, 2011.
- [14] The international convention for the prevention of pollution from ships. Technical report, International Maritime Organization, Protocolo, 1978.
- [15] K.M. Karri. Hull shape optimization for wave resistance using panel method. Master's thesis, Naval Architecture and Marine Engineering Department, University of New Orleans, 2010.
- [16] B. B. M. Kassab. Mesh generation on curved parametric surfaces based on a modified quadtree algorithm. 2009. PrePrint.
- [17] www.lua.org.
- [18] Antonio Carlos Oliveira Miranda and Luis Fernando Martha. Uma biblioteca computacional para geração de malhas bidimensionais e tridimensionais de elementos finitos. *Anais do XXI Ibero Latino Americano Sobre Métodos Computacionais para Engenharia*, 1:1–10, 2000.
- [19] <http://www.plmsolutionseds.com/products/parasolid>.
- [20] V.O. Rasmussen. Hull form modeling in conceptual design by assembly and modification of 3d hull modules. Master's thesis, Marine Systems Design Department, Norwegian University of Science and Technology-NTNU, 2009.
- [21] www.wamit.com.

Real-time visualization of Moebius transformations in space using Quaternionic-Bezier approach

Vytautas Karpavicius

Faculty of Mathematics and Informatics
Vilnius University, Lithuania
vytautas.karpavicius@mif.vu.lt

Rimvydas Krasauskas

Faculty of Mathematics and Informatics
Vilnius University, Lithuania
rimvydas.krasauskas@mif.vu.lt

ABSTRACT

Moebius transformations in space are much more sophisticated than the classical case on the plane, which has been well studied. We present a WebGL approach for visualization of Moebius transformations in 3-space by animating deformations of geometric objects composed of patches parametrized by Quaternionic-Bezier formulas. The idea is to represent Moebius transformations in a quaternionic form as well, and to use GPU shaders for transforming control points, weights, and normals, then seamlessly stitching patches with different levels of detail, and computing points on every patch. Finally, we demonstrate the main classes of Moebius transformations in space on several 3D objects including primitive shapes, Dupin cyclide patchworks, and Utah Teapot.

Keywords

Moebius transformation, Quaternionions, Quaternionic-Bezier, Visualization, GPU, WebGL, Shaders

1. INTRODUCTION

This paper was inspired by a wonderful video clip “Moebius Transformations Revealed” by Arnold and Rognes, which is available in various formats online [Arn09] (the theory behind is described in the paper [Arn08]) and some recent Quaternionic-Bezier surface constructions [Kra11].

Since our goal is to visualize Moebius transformations in 3-space, we cannot apply the aforementioned Arnold-Rognes approach directly. Indeed, for that one needs to show 3-sphere in 4-space. So we switched to the idea of animating deformations of familiar geometric objects in space using real time graphics. The concept of a Quaternionic-Bezier surface is the other important aspect of the proposed visualization method. Moebius transformations of the plane are usually identified with fractional-linear functions of complex variable, when complex numbers are treated as points on that plane (see e.g. [Arn08]). In the 3D case one can change complex numbers by quaternions and derive similar formalism based on 2×2 quaternionic matrices [Bis10].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

This approach in combination with ideas of [Gwy12] allows us to derive short original proofs of Theorems 1 and 2, which describe the main properties of Moebius transformations in 3-space.

Finally, we choose WebGL framework for implementation of our visualization method, since this modern technology enables us to deliver interactive real time 3D graphics in web environment.

The paper is organized as follows. We review definitions of quaternions, Quaternionic-Bezier formulas, and we describe the main properties of 3D Moebius transformations in Section 2. Section 3 is devoted to the visualization framework using WebGL. Examples of screenshots, showing the behavior of several 3D objects under Moebius transformations, are presented in Section 4. Finally, the conclusions are derived in Section 5.

2. USING QUATERNIONS

Algebra of Quaternions

We will use the algebra of quaternions \mathbf{H} with the standard basis $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$ and product rules:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -\mathbf{1}, \mathbf{ij} = \mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{ki} = \mathbf{j}.$$

Any quaternion $q = r\mathbf{1} + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ can be decomposed in its real part $\text{Re}(q) = r$ and its imaginary (vector) part $\text{Im}(q) = x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = \mathbf{v}$,

$$q = \text{Re}(q) + \text{Im}(q) = r + \mathbf{v}.$$

Reals \mathbf{R} and space \mathbf{R}^3 are identified with subsets in \mathbf{H} : $\{q \in \mathbf{H} : \text{Im}(q) = 0\}$ and $\mathbf{H}_0 = \{q \in \mathbf{H} : \text{Re}(q) = 0\}$.

Other useful notations for quaternions $q = r + \mathbf{v}$:

- conjugate $\bar{q} = r - \mathbf{v}$
- norm (length) $|q| = \sqrt{r^2 + \mathbf{v} \cdot \mathbf{v}}$
- inverse (if $q \neq 0$) $q^{-1} = \bar{q} / |q|^2$

If $|q| = 1$ then q has a *trigonometric form*

$$q = \cos \alpha + \sin \alpha \mathbf{u}, \quad \mathbf{u} \in \mathbf{H}_0, \quad |\mathbf{u}| = 1, \quad \text{and a square root of } q \text{ is defined}$$

$$\sqrt{q} = \cos \frac{\alpha}{2} + \sin \frac{\alpha}{2} \mathbf{u}. \quad (1)$$

Moebius Transformations in Space

Moebius (M) transformations in space are defined as compositions of inversions in space with respect to spheres. Alternatively, M-transformations can be generated by 4 elementary transformations in $\mathbf{R}^3 = \mathbf{H}_0$:

1. translation $x \mapsto x + a, a \in \mathbf{H}_0$,
2. scaling $x \mapsto \lambda x, \lambda \in \mathbf{R}$,
3. rotation about axis \mathbf{u} by angle 2α :
 $x \mapsto qxq^{-1}, q = \cos \alpha + \sin \alpha \mathbf{u}$,
4. inversion with a center in the origin and unit radius: $x \mapsto -x^{-1}$.

Note that the first 3 types of elementary transformations generate *Euclidean similarities*.

Arbitrary M-transformations can be represented by fractional-linear functions

$$F(x) = (ax + b)(cx + d)^{-1} \quad (2)$$

of quaternion variable x or by 2×2 matrices

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad a, b, c, d \in \mathbf{H},$$

using notations $F(x) = A * x$.

It is easy to check that usual multiplication of matrices corresponds to the composition of fractional-linear functions. The formula (2) defines transformation of all quaternions \mathbf{H} . The subgroup of all 2×2 matrices that define M-transformations of \mathbf{H}_0 is characterized in [Bis10], Theorem 11.1.

Elementary M-transformations correspond to the following matrices (here $a \in \mathbf{H}_0, \lambda \in \mathbf{R}, |q| = 1$):

$$\text{Tr}(a) = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}, \quad \text{Rot}(q) = \begin{pmatrix} q & 0 \\ 0 & q \end{pmatrix},$$

$$\text{Sc}(\lambda) = \begin{pmatrix} \lambda & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{Inv} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

Two theorems below are formulated following similar results in [Gwy12] about M-transformations in 4-space, but our proofs are different.

Theorem 1. For any given three distinct points $a_0, a_1, a_2 \in \mathbf{H}_0$ and another triple of distinct points $b_0, b_1, b_2 \in \mathbf{H}_0$ there exists an M-transformation F , such that $F(a_0) = b_0, F(a_1) = b_1, F(a_2) = b_2$.

If both triples of points are not collinear, F maps a plane going through points a_0, a_1, a_2 to another plane.

Sketch of the proof. Similar to the 2D case it will be convenient to extend our 3-space by adding the infinite point ∞ . Then we observe that M-trans-

formations that preserve ∞ are Euclidean similarities. Define transformation

$$H(a_0, a_1) = \text{Tr}(-a'_1) * \text{Inv} * \text{Tr}(-a_0),$$

$$a'_1 = \text{Inv} * \text{Tr}(-a_0) * a_1,$$

that maps the triple (a_0, a_1, a_2) to $(\infty, 0, a'_2)$. Similarly, $H(b_0, b_1)$ maps (b_0, b_1, b_2) to $(\infty, 0, b'_2)$, and it remains to find an appropriate transformation $R(a'_2, b'_2)$ from $(\infty, 0, a'_2)$ to $(\infty, 0, b'_2)$. This can be a composition of rotation and scaling (see (1))

$$R(a, b) = \text{Sc}(|q|) * \text{Rot}(\sqrt{|q|}|q|^{-1}), \quad q = ba^{-1}.$$

Finally, the composition

$$F = H(b_0, b_1)^{-1} * R(a'_2, b'_2) * H(a_0, a_1)$$

will map (a_0, a_1, a_2) to (b_0, b_1, b_2) .

Theorem 2. Let $a_0, a_1, a_2, a_3 \in \mathbf{H}_0$ be four distinct points, and let $b_0, b_1, b_2 \in \mathbf{H}_0$ be three distinct points. Then the set of the images $F(a_3)$ for all M-transformations F , such that $F(a_0) = b_0, F(a_1) = b_1, F(a_2) = b_2$ is either a single point (if points a_0, a_1, a_2, a_3 are co-circular), a line or a circle.

Sketch of the proof. According to the Theorem 1 one can suppose that $(a_0, a_1, a_2) = (b_0, b_1, b_2) = (\infty, 0, \mathbf{i})$. Then 3 points $\infty, 0, \mathbf{i}$ are fixed and we are looking for Euclidean similarity with 2 fixed points $0, \mathbf{i}$. There are two cases: $0, \mathbf{i}, a_3$ are collinear or not. In the first case all 4 points $\infty, 0, \mathbf{i}, a_3$ are on a line (i.e. points a_0, a_1, a_2, a_3 are co-circular) and we cannot move a_3 to any other position. In the second case the only possibility for a_3 is to rotate around the axis going through the both $0, \mathbf{i}$. Therefore, $F(a_3)$ can be any point of the particular circle (or line in general).

Quaternionic-Bezier (QB) Formulas

Here we remind some results published in [Kra11].

Circular Arc

Let p_0 and p_1 be two endpoints of a circular arc C in $\mathbf{R}^3 = \mathbf{H}_0$, and let q be some interior point on C . QB-curve of degree 1 defined by the formula

$$C(t) = (p_0 w_0 (1-t) + p_1 w_1 t) (w_0 (1-t) + w_1 t)^{-1}$$

with quaternionic control points p_0, p_1 and weights

$$w_0 = (q - p_0)^{-1}, \quad w_1 = (p_1 - q)^{-1},$$

defines a rational parametrization $[0,1] \rightarrow \mathbf{H}_0$ of C .

A tangent vector v_0 at the endpoint p_0 is

$$v_0 = (p_1 - p_0) w_1 w_0^{-1}. \quad (3)$$

Bilinear QB-surface patch

Let us define a *bilinear QB-surface patch* by the usual rational Bezier formula but with quaternionic control points p_{ij} and weights w_{ij} ($0 \leq s, t \leq 1$):

$$P(s, t) = \left(\sum_{i=0,1} \sum_{j=0,1} p_{ij} w_{ij} s_i t_j \right) \left(\sum_{i=0,1} \sum_{j=0,1} w_{ij} s_i t_j \right)^{-1},$$

where $s_0 = 1-s, s_1 = s, t_0 = 1-t, t_1 = t$.

Here we consider just two important cases: Dupin cyclide and Darboux cyclide (cf. [Kra11], [Pot12]).

Let p_0, p_1, p_2, p_3 be any 4 points on a circle in $\mathbf{R}^3 = \mathbf{H}_0$, and let $\mathbf{v}_1, \mathbf{v}_2$ be two orthonormal vectors, i.e. $|\mathbf{v}_1| = |\mathbf{v}_2| = 1$ and $\mathbf{v}_1 \perp \mathbf{v}_2$. Then there is a unique *principal Dupin cyclide patch* with corners in these points, and bounded by circular arcs with tangent vectors $\mathbf{v}_1, \mathbf{v}_2$ at the corner p_0 , which can be rationally parametrized by the formula $P(s, t)$, where double index notations are changed to single ones

$$\{00, 01, 10, 11\} \rightarrow \{0, 1, 2, 3\}.$$

Here weights w_i are computed by formulas:

$$w_0 = 1, \quad w_1 = q_{10}v_1, \quad w_2 = q_{10}v_1,$$

$$w_3 = \delta_{12}\delta_{03}^{-1}q_{31}w_1q_{20}w_2,$$

where

$$\delta_{ij} = |p_i - p_j| \in \mathbf{R}, \quad q_{ij} = (p_i - p_j)\delta_{ij}^{-1} \in \mathbf{H}_0.$$

A *Darboux cyclide patch* is the most general case of a bilinear QB-surface [Kra11]. For our purposes it will be enough to consider cases with unit weights and their images under M-transformations, when the correct weights will be automatically computed as explained below.

Moebius invariance

QB-curves and QB-surfaces are Moebius invariant: their image under a certain M-transformation has the same formula, where only control points and weights need to be changed $p \mapsto p', w \mapsto w'$:

$$p' = F(p) = (ax + b)(cx + d)^{-1}, \quad (4)$$

$$w' = FW(p, w) = (cp + d)w. \quad (5)$$

3. WEBGL FRAMEWORK

Here we describe a framework which enables us to visualize Moebius transformations in the web environment. Transforming, computing and rendering QB surfaces are very computationally intensive tasks. While it is possible to do it all in javascript, it would be terribly slow and would not allow us to represent real-time animations of such transformations. This framework offloads most of the computation tasks to the GPU which is well suited for such massively parallelizable computations [Bro13].

We have chosen to deliver visualization content in web environment so that it would be available to wider audience with ease of access.

GPU is reached through an API exposed by WebGL technology [Khr12], which is based on OpenGL ES 2.0 [Khr10]. Even if it does not allow to use the most modern GPU features, it is possible to achieve desired result solely through WebGL API.

An abstract outline of the framework algorithm:

1. Initialization
 - 1.1. Loading a model
 - 1.2. Preparing buffers
2. Computing surface points
 - 2.1. Transforming position, weights, normals

2.2. Estimating the level of details for patches

2.3. Mapping patches to batches

2.4. Computing batches

3. Rendering surface points

Surface points are recomputed every time the surface is transformed. Surface gets rendered when the camera has moved or user changes any of the display options (e.g. switches to wireframe mode).

Loading a Model

We start by loading a model from .obj file. The file format is adjusted to accommodate weights which are required by QB patches. Each weight is defined by a line starting with w and following 4 real numbers separated by spaces – representing 4 components of the quaternion in the order: x, y, z, r. Then each face references their weights by the index in adjusted face vertex definition format: position/uv/normal/weight.

Three textures are created to fit positions, normals and weights. Each face stores their vertex attributes separately in these textures (vertices are no longer shared among faces). Face vertices correspond to 4 subsequent pixels in textures. After this data has been uploaded to GPU, each face vertex object in javascript gets a reference to their respective data in textures, that is, a fetch coordinate is stored.

After loading the model, we search for adjacent faces. This is required later when patches of different LOD are stitched together. Each face gets references to their adjacent left, right, top and bottom faces.

Preparing Buffers

Buffers which do not depend on transformation state get precomputed and uploaded to GPU in advance. For every LOD level 7 buffers need to be precomputed. 4 of those will be used during the computation stage, other 3 – during the rendering stage. Those buffers will be described in surface point computation and rendering sections respectively.

Computing Surface Points

3.1.1 Render to Texture

To use GPU for arbitrary computation instead of just rendering images, we take advantage of the technique called render-to-texture. Instead of drawing to the screen, custom texture is attached to the framebuffer and rendered image gets stored in it. The data from the framebuffer then can be read back to the main memory and processed with the CPU. Alternatively the texture containing rendering results can be used in subsequent rendering passes as a data source.

A quad is drawn to cover entire texture. Quad vertices have attributes (0;0), (0;1), (1;0) and (1;1) which get interpolated and passed to the fragment shader. This interpolated parameter lets the fragment shader know which fragment it is working with and proceed with computation accordingly.

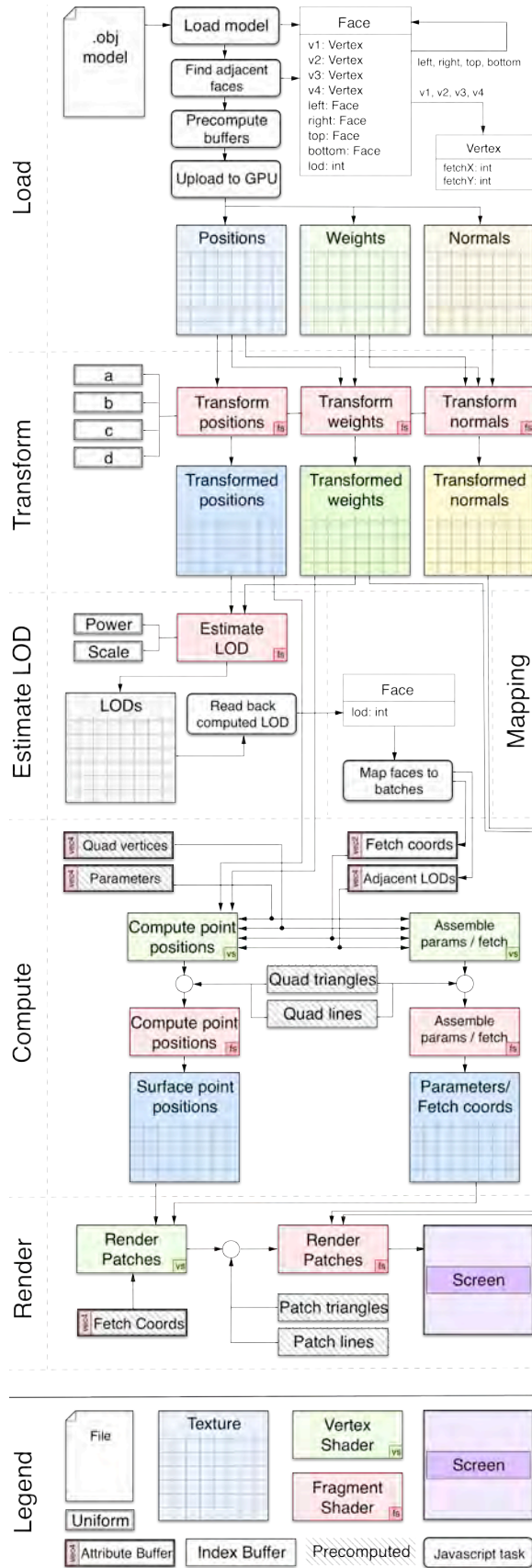


Figure 1. Diagram showing data flow across all stages in the framework.

3.1.2 Applying transformations

All Mobius transformations can be reduced to 4 quaternion coefficients a, b, c, d (see (2)). Those coefficients are then set as vec4 uniform parameters in 3 fragment shaders which transform positions, weights and normals of control points.

Formulas (4), (5) are used for transforming control points and weights. The transformation of normal n at point p_0 can be achieved by transforming to two points: p_0 and $p_1 = p_0 + n$. In general Moebius transformation of a line segment $[p_0, p_1]$ will be a circular arc. Therefore, to get the transformed normal we compute a tangent to that arc according to formulas (3)-(5):

$$n' = (F(p_1) - F(p_0))FW(p_1, 1)FW(p_0, 1)^{-1}.$$

Estimating Level of Detail

Moebius transformation can significantly deform the surface of the model. Our framework implements a feature to estimate dynamically the level of detail (LOD) for each patch. The more curved patches will get higher level of detail, the more surface points will be evaluated.

The level of detail for each patch is computed in a separate shader by taking transformed positions and weights of the patch. The results are then read back to the main memory so that javascript code could map patches to appropriate buffers.

One can choose different ways to estimate LOD. Note that it is not crucial to have mathematically exact formula to find the curvature of the patch. Our implementation takes into account two measures: the size of the patch L and the distance H between middle points of transformed patch f and flat plane on patch control points c .

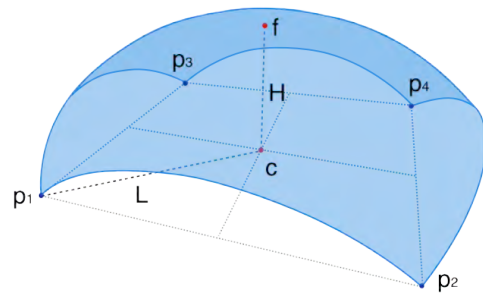


Figure 2. Measures used in estimating LOD.

The level of detail is computed by taking ratio H/L , raising it to the power p and then scaling by s . Constants p and s are passed to the shader as uniform parameters. They allow a user to fine tune surface LOD.

$$\text{LOD} = s \left(\frac{H}{L} \right)^p \quad (4)$$

Mapping Patches to Batches

The patches of the surface are computed and rendered in groups. A bunch of patches having the same level of detail gets layed out in the texture next to each other. Such texture is called a batch. In other words, a buffering mechanism for data sent to the GPU is created in order to achieve better performance.

All batches are of the size 256×256 . It gives us a total of 65536 (or 2^{16}) pixels in the texture. Later, when the batch is rendered, an index buffer of triangles (or lines) is used, where each index is of type `gl.UNSIGNED_SHORT`. Therefore a whole batch gets rendered in one draw call. There is no point to make bigger batches since we will not be able to render them at once anyway.

Minimum level of details is 2×2 . It takes only original control points without interpolating any additional internal points. One such batch can hold up to 16384 patches. Maximum available LOD – 256×256 . A patch with this LOD takes up the whole batch. So there are 8 different levels of details: 2, 4, 8, 16, 32, 64, 128 and 256.

Every time the model is transformed, we need to remap patches since their LOD may have changed. It is done by storing fetch coordinates of face vertices to the attribute buffer. The shader that is used to compute the batch will use these coordinates to fetch transformed positions and weights from textures that had got computed in the transform stage. At this point we also prepare LODs buffer of adjacent patches. Each vertex gets `vec4` attribute where components x, y, z, w store LODs of bottom, right, top and left adjacent patches. Finally, we increment the counter that stores the amount of patches this batch holds. If all patches of the same LOD do not fit in one batch, additional batches are created.

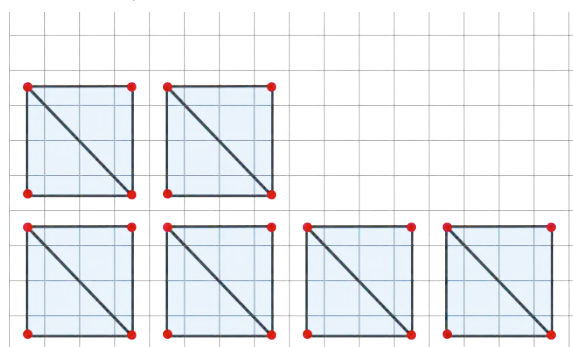


Figure 3. Example of 4×4 patches (quads) mapped to the batch texture.

Computing Batches

3.1.3 Batch buffers

A batch gets computed by drawing a bunch of quads to the texture. Each quad represents a patch. In order to draw those quads we need 6 buffers. 2 of them are

prepared during the mapping stage – a buffer of fetch coordinates and a buffer of adjacent LODs. The other 4 are precomputed in advance. Those include: quad vertex buffer, parameter buffer, triangle index buffer and lines index buffer.

The quad vertex buffer contains coordinates (x, y) in batch space. That is, they are integers from zero which refer to the pixels of the texture. For $\text{LOD}=4$, this buffer would look like: $(0,0), (3,0), (0,3), (3,3), (4,0), (7,0), (4,3), (4,7) \dots$

In the vertex shader these coordinates are transformed to homogeneous space using this function:

$$f(\vec{x}) = \begin{pmatrix} -1 \\ -1 \end{pmatrix} + \frac{1}{256} \left(2\vec{x} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \quad (5)$$

The scaling factor $1/256$ comes from the fact that we use 256 sized batches. As fragments are evaluated at their centers, we also need add vector $(1,1)$. This shifts quads so that their corners will be at the center of fragments.

The previous function can also be written as matrix, transforming quad vertices to homogeneous space. The quad vertex vector also needs to be expanded to 4 components vector: $(x, y) \rightarrow (x, y, 0, 1)$.

$$\begin{bmatrix} 1/128 & 0 & 0 & -255/256 \\ 0 & 1/128 & 0 & -255/256 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The parameter buffer holds a repeating sequence of $(0,0), (1,0), (0,1), (1,1)$ vectors that will act as s and t parameters in the equation of the bilinear QB-surface $P(s, t)$. These vectors are passed to fragment shader, where they will get interpolated for every fragment.

Finally, we have index buffers for drawing triangles and lines. The usage of triangles index buffer is straightforward – to connect vertices into primitives (quads). However, because of rasterization rules not all fragments get covered by triangles. A fragment is covered by a triangle if the center of that fragment is inside the triangle. If a fragment center happens to be exactly on triangle edge – the top-left rule is applied [Mic12]. It states that the fragment is rasterized if it is on the left edge (or the top, in case the edge is horizontal).

The bottom and right edges of the quad do not get rasterized. For this reason we also need lines buffer which is used for drawing edge lines of the patch.

3.1.4 Evaluating surface points

The evaluation of surface points is performed in the fragment shader. Bilinear interpolation of 4 control points is achieved using 3 `mix` instructions. Two of them interpolates in x axis (between 0-1 and 2-3 points). The third then interpolates in y axis between the results of the previous. This interpolation needs to

be performed for $p_i w_i$ and for w_i . These will give us the numerator and denominator of the bilinear QB-surface $P(s, t)$. The excerpt of fragment shader is given below:

```
vec4 a = mix(vPW[0], vPW[1], s);
vec4 b = mix(vPW[2], vPW[3], s);
vec4 nomin = mix(a, b, t);
a = mix(vW[0], vW[1], s);
b = mix(vW[2], vW[3], s);
vec4 denomin = mix(a, b, t);
gl_FragColor = qMult(nomin, qInv(denomin));
```

Note that premultiplied $p_i w_i$ and w_i of each control point are passed from vertex shader as varying parameters. Even though varying parameters are interpolated, we do not need that. They are used just as a way to pass data from the vertex shader. To cancel interpolation, each vertex in the quad has to compute the same values for the varyings.

The vertex shader receives fetch coordinates in an attribute buffer that was assembled during patch mapping. Using those coordinates, it can fetch transformed positions and weights of control points. It is not enough for a vertex to fetch its own position and weight. All 4 control points need to be fetched so that they could be passed to the fragment shader canceling unwanted interpolation.

Consider that current quad vertex has fetch coordinate (x, y) . Then all 4 fetch coordinates $f_i, i = 0, 1, 2, 3$, can be found using:

$$f_i = \left(\frac{x - \text{mod}(x, 4) + i}{y} \right) \quad (6)$$

Note that if we were working with triangle patches (instead of quad patches), this varying-cancellation mechanism would not be required. Every vertex would simply fetch its own attributes and pass them through varyings to the fragment shader where they would be already interpolated. However, the interpolation that occurs during rasterization, works only for triangles, that is why we need our own interpolation system for quads.

3.1.5 Stitching patches of different LODs

Dynamically computing patches of different level of detail allows us to render them with enough accuracy while keeping performance required for real-time animations. Due to this, patches with different LOD do not align perfectly and create gaps.

We have established a patch stitching mechanism which alters how border points of the patch surface are evaluated. The idea is to supply every patch with information about LOD of neighbouring patches. In this way the patch checks if neighbouring patch has lower LOD, and if this is the case, that means the points of the edge are snapped to the nearest points of the neighbour patch.

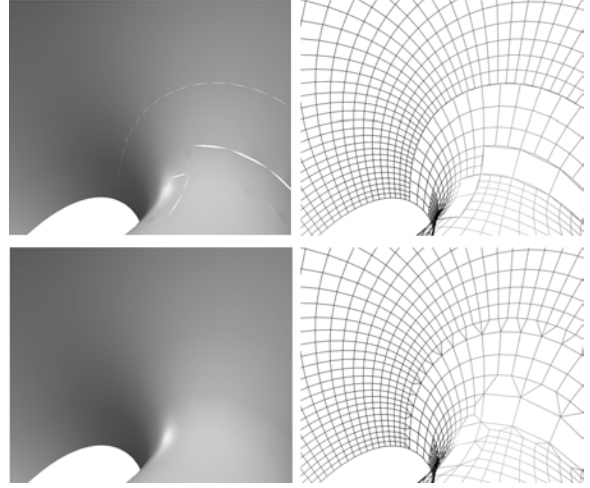


Figure 4. Surface without patch stitching (upper) and surface with patch stitching (lower).

Given that surface point evaluation coordinate (across X axis) is x and n is required LOD for the edge of this patch, evaluation coordinate can be adjusted using the following formula

$$x' = \text{round}(x(n-1)/(n-1)). \quad (7)$$

This correction has to be applied to the bottom ($y = 0$) and top ($y = 1$) edges. Similarly for the left ($x = 0$) and right ($x = 1$) edges the same correction is applied, except for different axis – Y .

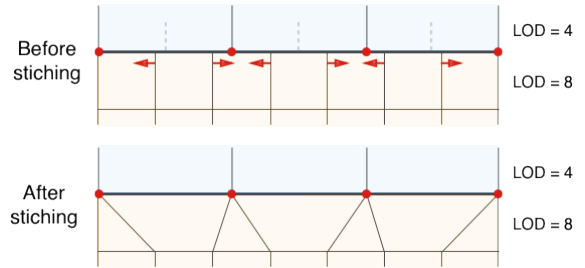


Figure 5. Patch edge vertices snapped to lower level of detail neighbouring patch.

Shader program does not need to know what the LOD of current patch is. It needs only 4 LODs for patch edges. This information is collected in javascript and passed to the shader as an attribute buffer.

Rendering Batches

Batches are rendered by fetching vertex positions from textures computed in the last stage. Fetch buffers for every different LOD are precomputed in advance. They contain (x, y) coordinates of vertices that make up patches. Two index buffers are also prepared for every different LOD batch. The lines index buffer is used for displaying a model in wireframe mode. The triangle index buffer is used for a solid mode. Alternatively, a user can choose to display only surface points. No index buffer is used then.

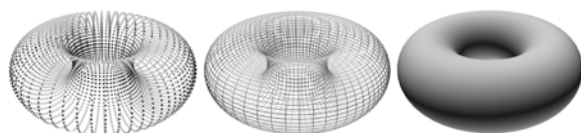


Figure 6. Different display modes: points, wireframe and solid.

While geometry of the surface is approximated to a certain level of detail, normals are computed exactly. They get evaluated per-fragment when rendering the final image which results in more accurate and better looking lighting.

For each batch auxiliary texture, containing fetch coordinates and parameters, is prepared. The texture components xy are used to store QB-patch parameter. The components zw store fetch coordinates. The sole purpose of this texture is to carry information that is available during computation phase to the rendering phase. This texture would not be required if we have chosen to compute normals together with positions in the batch computation stage. Positions/Normals textures would be passed from the compute stage to the rendering stage. However, this approach would only give us normals per-vertex.

Performance

The prototype of the framework was implemented and its performance was evaluated. Below there are the results of the visualization of various models. The evaluation was performed on the Nvidia 9400M GPU, Safari 6 browser. The results were collected while the 5 seconds animation of Clifford transformation was rotating model from 0 to 2π .

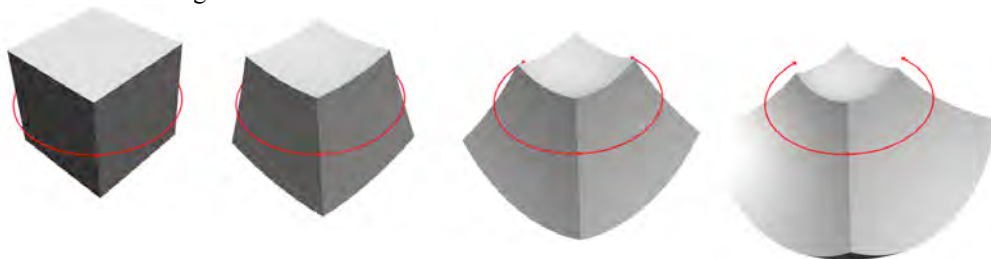


Figure 7. “Rotating” a cube about the fixed circle.

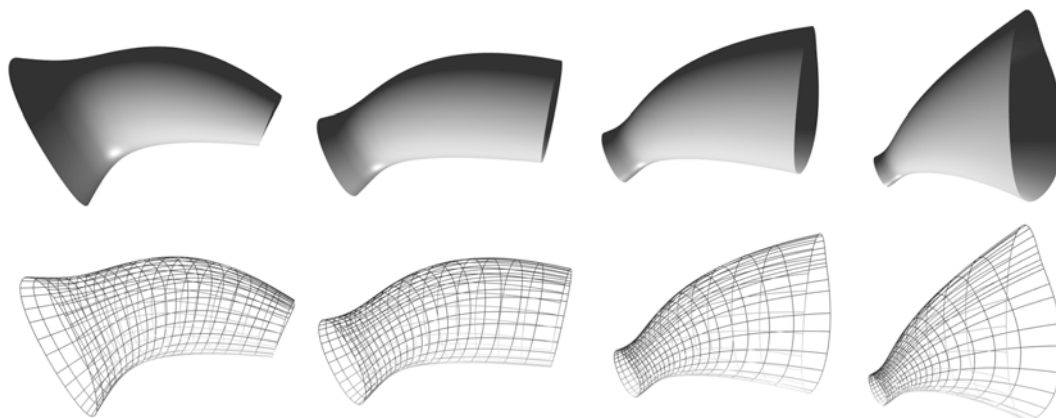


Figure 8. Deformation of a smooth Dupin cyclide patchwork (surface model by [Bo10]).

During transformation number of computed points varied because LOD of patches was changing.

Model	Patch count	Min points	Max points	Min FPS	Max FPS
Square	1	4	65536	39	74
Cube	6	24	82944	31	52
Sphere	4	4096	4096	45	76
Cone	2	2048	2048	47	84
Cylinder	2	2048	8192	49	77
Torus	4	1024	40960	36	55
Teapot	3305	13220	467648	12	16
Model1	150	21504	40512	35	56
Model2	96	22800	34176	38	58
Model3	56	37376	57344	28	65

Table 1. Performance of the framework.

The results show that the performance of all models except for the teapot was high enough to visualize transformations in real-time.

An alternative for this framework could be brute force approach: precompute points of all patches with high enough fixed LOD. Transformation would then be applied for all final points directly. Even though this might work well on modern GPUs, we would lose the structure of the surface. That is, it will no longer be possible to modify control points as well transformation parameters without recomputing data buffers.

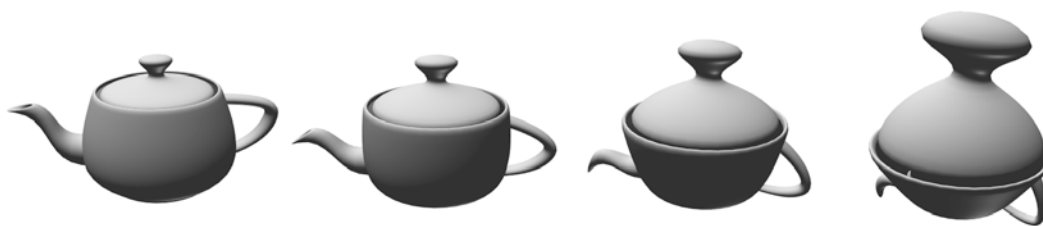


Figure 9. Clifford translation of Utah Teapot.

4. EXAMPLES

Using Theorems 1 and 2 one can control M-transformations by 3 or 4 points. Several examples are illustrated by figures:

Example 1. *Rotation about a circle* (Fig. 7): 3 points are fixed on the given circle and the 4-th point is a vertex of the cube and moves in circular orbit (cf. Theorem 2). Faces of the cube are deformed to spherical patches.

Example 2. *Hyperbolic transformation* (Fig. 8): 2 endpoints of the circular arc are fixed and the 3-rd point moves along that arc (see Theorem 1). The model is a smooth patchwork of Dupin cyclides, borrowed from [Bo10] (look for details in [Bo11]).

Example 3. *Clifford translation* (Fig. 9): M-transformation of Example 1 composed with Euclidean rotation along the circle going through the first 3 points. The model is the quad mesh of Utah Teapot.

An interactive web-based visualization tool is accessible at: <http://mif.vu.lt/~vyka6761/quaternionic>

5. CONCLUSIONS

A web-based approach for real time interactive visualization of Moebius transformations in 3-space was introduced. Both surface constructions and space transformations were presented in a uniform quaternionic setting based on Quaternionic-Bezier formulas. Original proofs of the main properties of Moebius transformations in 3-space were derived. The proposed WebGL framework was evaluated in the prototype implementation.

One possible extension of the proposed visualization method is related to increasing degrees of surface patches. Indeed, the formulas (4) and (5) used for the transformed control points and weights can be generalized for arbitrary degrees. Then, for example, one can apply Moebius transformations to arbitrary NURBS surfaces.

We hope this paper will not only be useful for better understanding of Moebius transformations in space but also will attract attention to new opportunities for shape modeling and animations.

6. ACKNOWLEDGMENTS

Our thanks to Pengbo Bo for allowing us to use models of smooth Dupin cyclide patchworks from his PhD thesis [Bo10].

7. REFERENCES

- [Arn08] Arnold D.N., Rognes J., Moebius Transformations Revealed, *Notices of the AMS* 55 (2008), 1226-1231.
- [Arn09] Arnold D.N., Moebius Transformations Revealed, *Webpage*, updated February 14, 2009, <http://www.ima.umn.edu/~arnold/moebius/>
- [Bo10] Bo P., Surface Fitting and Developable Surface Modeling, *PhD Thesis, The University of Hong Kong*, 2010.
- [Bo11] Bo P., Pottmann H., Kilian M., Wang W., Wallmer J., Circular arc structures, *ACM Transactions on Graphics* 30 (2011), #101, 1-11. <http://www.geometrie.tugraz.at/wallner/cas.pdf>
- [Gwy12] Gwynne E., Libine M., On a Quaternionic Analogue of the Cross-Ratio, *Advances in Applied Clifford Algebras* 22 (2012), 1041-1053.
- [Bis10] Bisi C., Gentili G., Moebius Transformations and the Poincare Distance in the Quaternionic Setting, *Indiana University Mathematics Journal* 58 (2010), 2729-2764.
- [Kra11] Krasauskas R., Zube S., Bezier-like parametrizations of spheres and cyclides using geometric algebra, in: Guerlebeck, K. (Ed.), *Proceedings of 9-th International Conference on Clifford Algebras and their Applications in Mathematical Physics*, 2011, Weimar, Germany. <http://www.mif.vu.lt/~rimask/old/pdf/Bezier-like.pdf>
- [Pot12] Pottmann H., Shi L. and Skopenkov M., Darboux cyclides and webs from circles, *Computer Aided Geometric Design* 29 (2012), 77-97.
- [Khr10] The Khronos Group. OpenGL ES Common Profile Specification. Version 2.0.25, November 2, 2010. http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf
- [Khr12] The Khronos Group. WebGL Specification, Version 1.0.1, 2012. <https://www.khronos.org/registry/webgl/specs/1.0.1/>
- [Bro13] Brodtkorb A.R., Hagen T.R., Sætra M.L., GPU Programming Strategies and Trends in GPU Computing, *Journal of Parallel and Distributed Computing* 73 (2013), 4-13.
- [Mic12] Microsoft Developer Network, Rasterization Rules, 2012. [http://msdn.microsoft.com/en-us/library/windows/desktop/cc627092\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx)