ISSN 1213-6972 *Volume 20, Number 2, 2012*

Journal of WSCG

An international journal of algorithms, data structures and techniques for computer graphics and visualization, surface meshing and modeling, global illumination, computer vision, image processing and pattern recognition, computational geometry, visual human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

Vaclav Skala – Union Agency

Journal of WSCG

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia
Faculty of Applied Sciences
Univerzitni 8
CZ 306 14 Plzen
Czech Republic
http://www.VaclavSkala.eu

Managing Editor: Vaclav Skala

Printed and Published by:
Vaclav Skala - Union Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: ***ISSN 1213 – 6972***
CD ROM: ***ISSN 1213 – 6980***
On-line: ***ISSN 1213 – 6964***

Journal of WSCG

Vol.20, No.2

Contents

Real-time Volume Rendering and Tractography Visualization on the Web

John Congote[1], Esther Novo, Luis Kabongo
Vicomtech Research Center
Donostia - San Sebastian, Spain
jcongote,enovo,lkabongo@vicomtech.org

Dan Ginsburg
Children's Hospital
Boston, United States
dginsburg@upsamplesoftware.com

Stephan Gerhard
Institute of Neuroinformatics
Uni/ETH Zurich, Switzerland
connectome@unidesign.ch

Rudolph Pienaar
Harvard Medical School
Boston, United States
Rudolph.Pienaar@childrens.harvard.edu

Oscar E. Ruiz
[1]Universidad EAFIT
Medellin, Antioquia
oruiz@eafit.edu.co

ABSTRACT

In the field of computer graphics, *Volume Rendering* techniques allow the visualization of 3D datasets, and specifically, Volume Ray-Casting renders images from volumetric datasets, typically used in some scientific areas, such as medical imaging. This article aims to describe the development of a combined visualization of *tractography* and *volume rendering* of brain T1 MRI images in an integrated way. An innovative web viewer for interactive visualization of neuro-imaging data has been developed based on *WebGL*. This recently developed standard enables the clients to use the web viewer on a wide range of devices, with the only requirement of a compliant web-browser. As the majority of the rendering tasks take place in the client machine, the effect of bottlenecks and server overloading are minimized. The web application presented is able to compete with desktop tools, even supporting high graphical demands and facing challenges regarding performance and scalability. The developed software modules are available as open source code and include MRI volume data and tractography generated by the Diffusion Toolkit, and connectivity data from the Connectome Mapping Toolkit. Our contribution for the Volume Web Viewer implements early ray termination step according to the tractography depthmap, combining volume images and estimated white matter fibers. Furthermore, the depthmap system extension can be used for visualization of other types of data, where geometric and volume elements are displayed simultaneously.

Keywords

WebGL, Volume Rendering, Ray Casting, DVR, dMRI

1 INTRODUCTION

Three-dimensional data can be found in several scientific fields, coming from simulation, sampling or modeling processes. Regarding the biomedical scope, several scanning techniques, such as magnetic resonance (MRI) or computerized tomography (CT), are used for storing body imaging samples as volumetric datasets formed by groups of parallel slices, where the term volumetric dataset refers to a scalar field. These datasets are usually visualized in three dimensions in order to facilitate specialists to interpret information.

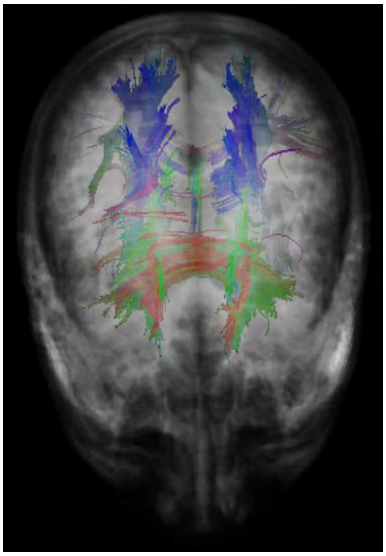


Figure 1: Combined visualization of volume rendering and tractography information on the web

Visualization of medical volumetric datasets can suitably be performed by the use of *Direct Volume Rendering* algorithms. These methods show important characteristics of datasets, even though rendering is not usually photo-realistic.The problem addressed in this paper is the visualization of tractography information obtained from dMRI (diffusion MRI) together with volume data corresponding to MRI or CT images.

In order to represent the volumetric datasets, volume rendering techniques allow the visualization of all inner characteristics of volumes at once, by projecting data into 2D images, according to the corresponding position of a virtual camera. The main idea of the ray-casting algorithm is to launch rays from the camera into the volume, calculating the volume rendering integral along the rays. Thus, in this method, the colour and opacity of each pixel in the final image is evaluated by launching a ray in the scene from the view position, sampling the volume at discrete points along the ray and accumulating the contribution of each sample.

Our contribution is an implementation of a web rendering system for medical images, which integrates volume rendering and geometric objects within a compliant WebGL browser, based on the volume ray casting algorithm and built on previous developments [CSK11]. Due to the technology limitations of Webgl, the improvements developed allow us to create a web application for combined visualization of volume rendering and tractography, as shown in Figure 1, being able to compete with desktop tools, supporting high graphical demands and facing challenges regarding performance and scalability.

The article is organized as follows. Section 2 presents the work related to this article, including a description of volume rendering techniques, visualization of medical images and geometry intersection. The methodology of the developed work is explained in Section 3. Then, the results accomplished are presented, and finally, Section 5 states the conclusions and future developments.

2 RELATED WORK

2.1 Volume Rendering

In computer graphics, Ray Casting is a well known direct volume rendering technique that was designed by Kajiya and Herzen [KVH84] as one of the initial developments in this area. Traditionally, three dimensional objects have been created by using surface representations, drawing geometric primitives that create polygonal meshes [Lev88], hence provoking the loss of information from one dimension.

Further developments [DCH88] accomplished the mathematical modeling of the ray casting process, based on the light's behaviour equations. Thus, the volume rendering integral was defined. A comparative between different direct volume rendering algorithms, such as Texture Mapping, Ray Casting, Splatting or Shear Warp, was presented [MHB00]. Ray casting is a flexible algorithm that allows the implementation of acceleration methods, such as Empty Space Skipping [KW03] or *Early Ray Termination*. Early ray termination is an optimization process that establishes certain limitations in the volume, so that the samples encountered after them do not contribute to the value of the pixel.

Ray casting suitably fits GPUs' operating mode [Sch05], because of the independence of each ray that is launched to the scene, making this algorithm highly parallelizable and allowing the exploitation of GPUs' parallel architecture. For GPU ray casting, the volume element is stored in the GPU memory as a 3D texture and a fragment shader program is used in order to implement the ray casting algorithm.

A quality evaluation model was developed for comparing the different Direct Volume Rendering techniques [BBD07]. These methods handle a higher amount of data than surface rendering techniques, therefore, the complexity of the algorithms is increased, as well as the necessary rendering time [Bru08]. Optimized volume rendering methods avoid empty spaces by introducing a volume proxy geometry [MRH08].

Web 3D Rendering

The use of the recently released WebGL standard [Mar11] leads to new methods for web 3D visualization, where most part of the computational processes are performed in vertex and fragment shaders that run on the GPU hardware. WebGL is a software library that enables HTML5-based browsers to identify clients' graphics hardware. HTML5, the latest Internet standard propose, provides native elements for audio and video. WebGL consists of a low-level imperative graphic programming API based on OpenGLES 2.0 for Javascript that enables flexibility and exploits the characteristics of advanced graphics cards. Due to the constant improvement of the performance of Javascript interpreters, the management of scene elements behaves similarly to the ones obtained by using natively compiled languages. Moreover, some WebGL extensions have been implemented in order to achieve a friendly interaction, such as SpiderGL [DBPGS10].

Several standards and proprietary solutions are currently being developed in order to fulfil the necessity of moving 3D visualization into the web [BA01], such as X3D, a standard derived from VRML that stores 3D information in a scenegraph format using XML (Extensible Markup Language). This model has been implemented in a declarative form, as an extension of HTML; X3DOM presents a framework for integrating

X3D nodes into HTML5 DOM content [BEJZ09] and other alternatives have also been developed, e.g. XML3D [SKR10]. Finally, there is a standardization for X3D in the MedX3D volume rendering model [JAC08, PWS11].

2.2 Visualization of Medical Images

Medical visualization is a challenging scientific field because interpretation of images may lead to clinical intervention. Therefore, quality and fast interactive response are important features in this domain. Remarkable advances have occurred in medical imaging technology and applications in the past few years, supporting the possibility of sharing imaging data online across clinical and research centres and among clinicians and patients. The development of these kind of applications is influenced by connectivity, security and resources' heterogeneity concerns.

On-server rendering can be considered a partial solution for Medical Imaging [BM07]. Moreover, several web implementations for volumetric visualization have already been presented [JAC08], although many of these solutions require third party systems to allow visualization or their scalability is limited by the rendering server.

As medical volumetric imaging requires high fidelity and high performance, several rendering algorithms have been analyzed, leading to thread- and data-parallel implementations of ray casting [SHC09]. Thus, architectural trends of three modern commodity parallel architectures are exploited: multi-core, GPU, and Intel Larrabee. Other approaches describe the development of web-based 3D reconstruction and visualization frameworks for medical data [SAO10]. Such applications based on X3D technology allow extending cross-platform, inter-application data transfer ability. Several applications have been implemented using web 3D rendering techniques, for example, evaluation systems at the educational level [Joh07] or medical training simulations [JROB08].

dMRI

Diffusion Magnetic Resonance Imaging (dMRI) relies on the visualization of water diffusion using data from MRI. Diverse methodologies have been presented over the last years and can be classified into two categories: Image based and Object based techniques. The first methodology divides the space in voxels and the assigned colour represents the principal diffusion direction [MAA03]. However, tracks can not be easily identified since no segmentation of the visualization is performed, and therefore direction information is difficult to observe since voxel colour mapping is not one-to-one, *i.e.*, different directions might be represented by the same colour. Otherwise, in object based methodologies, objects, such as ellipsoids and lines, are used together with colour mapping in order to enhance visualization and give a direction sense to the representation.

Visualization of brain white matter cortical tracks is one of the most important applications of dMRI, since it allows to non-invasively visualize white matter anatomy, and detecting of anomalies [NVLM07, GKN11]. Tractography, which refers specifically to the representation of the white matter tracks based on the water diffusion information, employs lines to represent the diffusion direction and to visualize the white matter paths. In general, lines are generated using randomly distributed seed points; together with the principal diffusion information and a prescribed interval of time, the different paths are generated. However, this representation becomes dependent on the amount and location of seed points to correctly visualize tracks [EKG06] because erroneous connections might be produced between tracks due to the existing error in data. Incorrect visualization of branching of tracks is another drawback, since only one path is generated per each seed point.

Probabilistic methodologies have been proposed [EKG06] to represent branching of white matter tracks, in which secondary seed points are included in regions in which branching is assumed. Therefore, a denser visualization is performed in those regions. An algorithm was proposed for path visualization [RSDH10], in which the different global paths are simplified by one simple curve, clustering the different paths and then using average curves to obtain one simple curve that summarizes each cluster.

2.3 Geometry Intersection

The application described in this article requires representing volume rendering and tractography together, *i.e.*, both volumetric and polygonal data have to be displayed in the same scene. There are several models for combining polygonal geometry and volume rendering. Some methods identify the intersection between rays launched in the volume rendering process and geometry [SMF00]. This technique can be optimized by creating octrees for dividing the geometric space and prove intersections correctly.

Other models try to achieve a correct visibility order for the intersections between volume and geometry [HLSR09]. Geometry has to be rendered in the first place to correctly look at the intersections of the geometry and the volume. Besides, parts that are occluded by the geometry should not contribute to the final image, not performing any ray casting at all. In order to achieve this feature, rays should terminate when they hit a polygonal object, accordingly modifying the ray length image if a polygonal object is closer to the view point than the initial ray length.

3 METHODOLOGY

In our project, the results of the Connectome Mapper are directly loaded in the browser using WebGL and JavaScript. The FreeSurfer cortical surface reconstruction binary files are loaded and processed in JavaScript and converted to WebGL vertex buffer objects for rendering. The surfaces are overlaid with per-vertex curvature values computed during the FreeSurfer processing stream. The tractography data is likewise parsed in the JavaScript code and rendered as line primitives coloured based on direction. Finally, the structural network itself is converted to JSON (JavaScript Object Notation) as an offline preprocess and loaded into the browser using JavaScript. The networks are visualized in 3D along with the fiber tracts and volumes enabling exploration of connectivity information in real-time.

The work described in this paper has been developed using volume ray casting, a widely used algorithm for generation of 2D representations from three dimensional volumetric datasets. The obtained images are 2-dimensional matrices $I : [1,h] \times [1,w] \rightarrow \mathbb{R}^4$ (w: width and h: height, both in pixels). Each pixel is represented by a colour expressed by a four-tuple of red, green, blue and alpha real-valued components, $(R,G,B,A \in [0,1])$.

An entire volume is represented by a 3-dimensional array of real values $V : [1,H] \times [1,W] \times [1,D] \rightarrow [0,1]$ (H: Height, W: Width, D: Depth of the represented volume, all of them in positive integer coordinates). Therefore, $V(x,y,z) \in [0,1]$. The projection model used in this work is called pin-hole camera [HZ03]. The pin-hole camera technique uses intrinsic $K \in M_{3\times4}$ and extrinsic $R \in M_{4\times4}$ real-valued matrices in order to project every 3D point $p \in \mathbb{P}^3$ onto a 2D point $p' \in \mathbb{P}^2$.

The volume ray casting algorithm defines the colour for each pixel (i,j) in the image, which is also known as projection screen, I, according to the values of a scalar field $V(x,y,z)$. This scalar field is associated to the points (x,y,z) reached by rays that are originated at a certain pixel or camera, represented as C in Figure 2. A cuboid geometry is generated with coordinates $(0,0,0)$ to $(1,1,1)$. This cube represents the boundary established for the volumetric dataset. Each ray intersects with the cuboid volume V at points $p_{(i,j)}(x,y,z)$ and $q_{(i,j)}(x,y,z)$, which represent the input and output coordinates of the ray into and out from the volume, respectively.

Then, each obtained ray pq is equi-parametrically sampled. For every sampled point $s(x,y,z)$ over the ray, an approximation of the scalar field $V(s)$ is calculated, commonly by using trilinear interpolation. The sampled points also influence the colour of the originating pixel, due to the use of a composition function (Equations 1-4), where the accumulated colour A_{rgb} is the colour of the point s in the volume V, and A_a is the transparency component of the pixel, which has a value of 1 at the end of the rendering process. Given a certain set of coordinates (x,y,z) in the volume and a ray step k, V_a is the scalar value of the volume V, V_{rgb} is the colour defined by the given transfer function V_a, S represents the sampled values over the ray and O_f, L_f are the general Opacity and Light factors.

$$S_a = V_a \times O_f \times \left(\frac{1}{s}\right) \tag{1}$$

$$S_{rgb} = V_{rgb} \times S_a \times L_f \tag{2}$$

$$A^k_{rgb} = A^{k-1}_{rgb} + \left(1 - A^{k-1}_a\right) \times S_{rgb} \tag{3}$$

$$A^k_a = A^{k-1}_a + S_a \tag{4}$$

In the ray casting process performed in this work, geometry G is formed by a set of segment lines L (although G could also be represented as a set of points P or triangles T). Each segment L is defined by two points in the space. Lines are projected through projection matrices onto a different image, where the values of colour (r,g,b,a) and depth $(depth)$ are defined for each pixel (x,y).

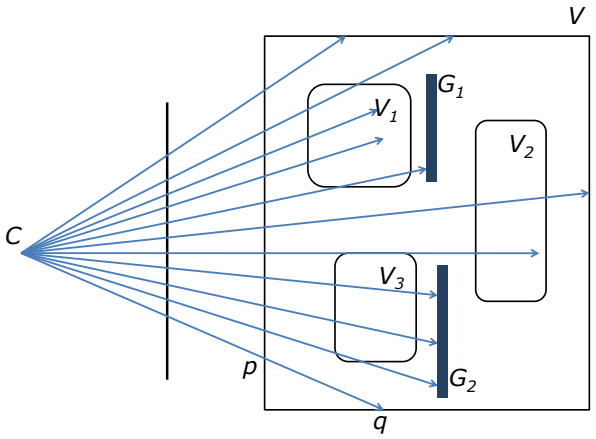


Figure 2: 2D representation of the ray casting algorithm performance (types of ray termination)

Each pq ray traverses the cuboid volume V, where both volume elements V_i and geometries G_i are rendered in the same process by modifying the early ray termination method, as depicted in Figure 2. This technique checks the alpha value for each sample of the transparency colour of the ray. If the value V_a is equal to 1, which means that the ray has reached the final colour, the remaining steps of the ray are not evaluated. Rays might terminate due to several reasons: when encountering a very dense volume (such as V_1 in fig. 2), when intersecting with a geometric element (*e.g.* with G_1) or when exiting the boundary cube, at point q.

The early ray termination model is also used to check the length of the ray and compare it to the depthmap of the figure. In conclusion, a projection of the geometry is

obtained, as well as the colour and depth for each pixel in the image. This information can be compared to the length of the ray, terminating the ray when the alpha value is 1 or when the depth is equal to the geometry depth.

4 RESULTS

This section describes the accomplished implementation of a real-time web viewer for both direct volume rendering and tractography visualization. This work is based on the WebGL standard and performs the ray casting algorithm with an early ray termination optimization.

4.1 Tractography

The Connectome Mapper [GDL11] is a publicly available software that provides a pipeline to automatically generate structural networks from raw dMRI data of the brain. Gray and white matter segmentations are obtained by processing T1 MPRAGE MRI using the Freesurfer set of tools. The Diffusion Toolkit is used later for reconstruction. A deterministic streamline algorithm is used to obtain tractography, by generating fiber tracts of the same subject. For cortical and subcortical regions of interest, a parcellation is performed. Finally, these datasets are coregistered and a network is generated by weighting the connectivity between regions based on the fiber tracts [GGCP11].

4.2 Data Processing and Volume Interpolation

For the developed work, all the slices that correspond to a certain volume are composed into a single image, as shown in Figure 3. This image is generated by placing slices in a matrix configuration as a preprocessing step of the rendering algorithm. The size of the image stored in GPU memory could range from 4096×4096 on a PC (which can contain up to 256^3 volume) to $1024x1024$ on other devices (which can contain up to $128 \times 128 \times 64$). The screen resolutions being reduced on mobile devices it seems reasonable to scale down or even crop the volumes original dimensions in order to match the maximum GPU available memory.

In medical imaging, the sample bit depth is usually higher than 8 bits per pixel. This is a drawback that has to be handled for the development of web applications, where commonly supported formats are limited to 8 bits per sample. In the described experiment, information from medical datasets was reduced to 8 bits per sample.

Identification of Ray Coordinates

According to the ray casting algorithm, the displayed colours of the boundary cuboid geometry represent

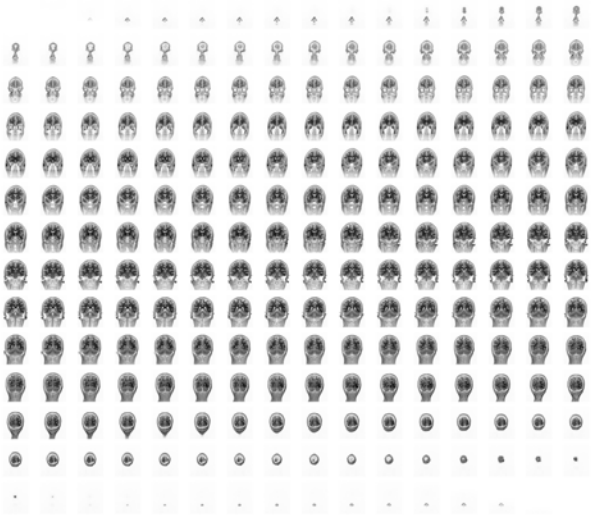


Figure 3: Brain dataset in mosaic form, read by the shader

the coordinates at each point (x,y,z). Coordinates are stored as r,g,b colour components for each pixel. Then, the cube can be rendered in the scene from the desired view point. In order to achieve volume visualization, several steps are followed in the rendering process. First of all, the rendering of the colour cube is performed according to the depth function change.

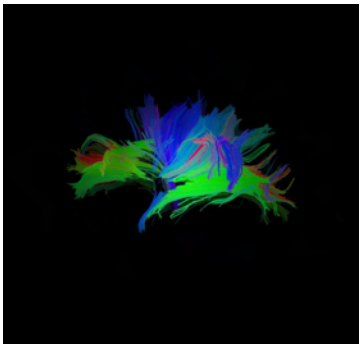
Taking this into account, rays are defined for each point of the cube, starting at the front faces, where the virtual camera is located, and ending at the back region. The colour of every point of the cube represents the exact coordinates of the ray for each pixel in the image. The colour information is stored as 24 bit RGB values. The range of values that can be represented may seem small or imprecise for large images, but colour interpolation provides precision enough for ray coordinates. The depth information is stored in different buffers in order to obtain the corresponding depth value for each ray. Finally, the geometry is rendered and the colour and depth buffers are stored to be processed in the volume shader.

4.3 Visualization

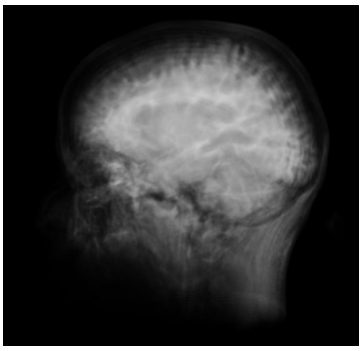
The previously presented GPU implementation of volume rendering based on WebGL was used to develop a real-time online tractography and volume rendering viewer, accordingly to Table 1, proving this standard to be a valid technology for real-time interactive applications on the web. The results shown in the table below were accomplished when interacting with the web viewer from several computers, using the same web browser (*Chrome*) and the same number of steps, 50. For every graphic card tested, the application can be completely considered to have a real-time behaviour.

Graphic card model	Frame rate
NVidia GeForce GTX480	60 fps
NVidia GeForce GTX285	60 fps
NVidia 9600GT	28 fps
NVidia Quadro FX 3800M	20 fps
NVidia Quadro FX 880M	15 fps

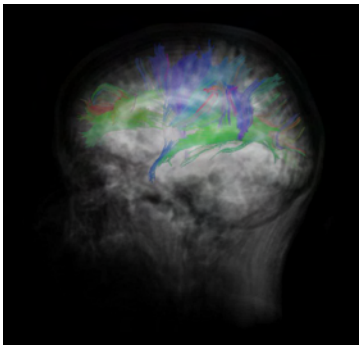
Table 1: Performance of the developed viewer for different graphic cards, using Chrome as web browser, the number of steps equal to 50



(a) Tractography



(b) Volume Rendering



(c) Combined visualization

Figure 4: Tractography, volume rendered image of brain T1 MPRAGE MRI and combined visualization on the web

In the developed work, the web viewer shows tractography information obtained from dMRI in the first place, represented in Figure 4(a). These organized fiber tracks in the white matter connect various cortical regions to each other. The tractography is represented using WebGL line primitives, where each fiber track is rendered by a set of points. The colour is assigned based on the absolute value of the unit vector pointing in the direction from the start point to the end point of the tract. The length value of each tract is stored in a per-vertex attribute together with the position and colour. The minimum tract length value is placed in a uniform variable in the vertex shader. The vertex shader determines whether the tract is longer than the minimum length to render. The entire tractrography set for the brain is efficiently rendered using a single draw call with one vertex buffer object. Thus, no dynamic geometry generation is performed in JavaScript.

Direct volume rendering of MRI data (Figures 4(b)) is developed simultaneously with the tractography. The volume renderer loads the MRI dataset from the server into a tiled 2D texture. Then, ray-tracing is performed in the shader in order to obtain the volume rendering. This implementation of a volume rendering system for the Web is based on the Volume Ray-Casting algorithm. Since the algorithm is implemented in WebGL, the reached visualization speed is similar to native applications, due to the use of the same accelerated graphic pipeline. The algorithm simulates 3D data by using a 2D tiling map of the slices from the volume maintaining trilinear interpolation and runs entirely in the client.

In the developed Web viewer, shown in Figure 5, the tractography and the volume rendering from brain MRI data can be represented separate or simultaneously, as depicted in Figures 4(c). Several features can be modified at runtime, by adjusting the provided sliders. Tractography's position can be changed according to the three main axes and fiber tracks can be seen more clearly by reducing the volume opacity. Finally, the minimum tract length can also be modified.

5 CONCLUSIONS AND FUTURE WORK

This paper describes the successful implementation of remote visualization of medical images based on WebGL[1]. Interaction with remote medical images was limited by many technical requirements, but the emergence of recent standards such as WebGL and HTML5 allow the development of applications that enable clients to access images without downloading them, maintaining

[1] http://www.volumerc.org/demos/brainviewer/webgl/brain_viewer/brain_viewer.html

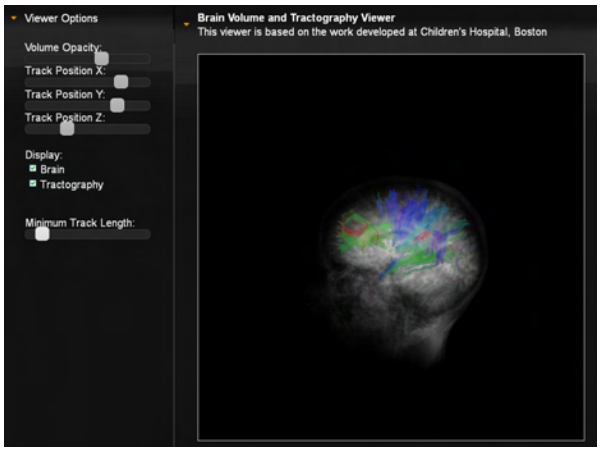


Figure 5: Volume rendering and tractography web viewer (sliders available for configuration)

data in a secure server and being able to perform functions, *e.g.* registration, segmentation, *etc.*, in a web context. These technologies empower web browsers to handle 3D graphics naturally. Thus, modern browsers support a wide range of applications, from simple rendering of two dimensional images to complex manipulation of 3D models.

The achieved visualization of volume rendering and tractography on the web, used for the implementation the presented viewer (shown in Figure 5), has demonstrated the capabilities of complex volume rendering visualization in web browsers, as well as the potential of WebGL for interactive visualization of neuroimaging data. Combined representation of volume rendering of brain T1 MRI images and tractography in real time has been accomplished. The main strength of the WebGL standard used here is the ability to provide efficient access to GPU rendering hardware with no special client-side software requirements, except for a compatible browser. Thus, this platform has great potential for imaging tools, particularly those providing web-based interfaces for automatic pipelining of image data processing.

In the work explained herein, the early ray termination algorithm was modified in order to combine volume and geometric elements in a seamless way. Thus, the developed software modules, which are available as open source code, successfully implement early ray termination step according to the tractography depthmap, performing a combination between volume images and estimated white matter fibers.

6 ACKNOWLEDGMENTS

This work was partially supported by CAD/CAM/CAE Laboratory at EAFIT University and the Colombian Council for Science and Technology -COLCIENCIAS-. Everyone who has contributed to this work is also gratefully acknowledged.

7 REFERENCES

[BA01] Johannes Behr and Marc Alexa. Volume visualization in vrml. In *Proceedings of the sixth international conference on 3D Web technology*, Web3D '01, pages 23–27, New York, NY, USA, 2001. ACM.

[BBD07] Christian Boucheny, Georges-Pierre Bonneau, Jacques Droulez, Guillaume Thibault, and Stéphane Ploix. A perceptive evaluation of volume rendering techniques. In *Proceedings of the 4th symposium on Applied perception in graphics and visualization*, APGV '07, pages 83–90, New York, NY, USA, 2007. ACM.

[BEJZ09] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[BM07] Bojan Blazona and Zeljka Mihajlovic. Visualization service based on web services. *29th International Conference on*, pages 673–678, 2007.

[Bru08] S. Bruckner. *Efficient Volume Visualization of Large Medical Datasets: Concepts and Algorithms*. VDM Verlag, 2008.

[CSK11] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 137–146, New York, NY, USA, 2011. ACM.

[DBPGS10] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. Spidergl: a javascript 3d graphics library for next-generation www. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 165–174, New York, NY, USA, 2010. ACM.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 65–74, New York, NY, USA, 1988. ACM.

[EKG06] H.H. Ehricke, U. Klose, and W. Grodd. Visualizing mr diffusion tensor fields by dynamic fiber tracking and uncertainty mapping. *Computers & Graphics*, 30(2):255–264, 2006.

[GDL11] S. Gerhard, A. Daducci, A. Lemkaddem, R. Meuli, J.P. Thiran, and P. Hagmann. The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes. *Frontiers in Neuroinformatics*, 5, 2011.

[GGCP11] Daniel Ginsburg, Stephan Gerhard, John Edgar Congote, and Rudolph Pienaar. Realtime visualization of the connectome in the browser using webgl. *Frontiers in Neuroinformatics*, October 2011.

[GKN11] A.J. Golby, G. Kindlmann, I. Norton, A. Yarmarkovich, S. Pieper, and R. Kikinis. Interactive diffusion tensor tractography visualization for neurosurgical planning. *Neurosurgery*, 68(2):496, 2011.

[HLSR09] Markus Hadwiger, Patric Ljung, Christof R. Salama, and Timo Ropinski. Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, pages 1–166. ACM, 2009.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition, 2003.

[JAC08] N W John, M Aratow, J Couch, D Evestedt, A D Hudson, N Polys, R F Puk, A Ray, K Victor, and Q Wang. Medx3d: Standards enabled desktop medical 3d. *Studies In Health Technology And Informatics*, 132:189–194, 2008.

[Joh07] Nigel W. John. The impact of web3d technologies on medical education and training. *Computers and Education*, 49(1):19 – 31, 2007. Web3D Technologies in Learning, Education and Training.

[JROB08] Yvonne Jung, Ruth Recker, Manuel Olbrich, and Ulrich Bockholt. Using x3d for medical training simulations. In *Web3D '08: Proceedings of the 13th international symposium on 3D web technology*, pages 43–51, New York, NY, USA, 2008. ACM.

[KVH84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18:165–174, January 1984.

[KW03] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988.

[MAA03] Y. Masutani, S. Aoki, O. Abe, N. Hayashi, and K. Otomo. Mr diffusion tensor imaging: recent advance and new techniques for diffusion tensor visualization. *European Journal of Radiology*, 46(1):53–66, 2003.

[Mar11] Chris Marrin. *WebGL Specification*. Khronos WebGL Working Group, 2011.

[MHB00] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90. Citeseer, 2000.

[MRH08] Jörg Mensmann, Timo Ropinski, and Klaus Hinrichs. ccelerating volume raycasting using occlusion frustums. In *IEEE/EG Volume and Point-Based Graphics*, pages 147–154, 2008.

[NVLM07] P.G.P. Nucifora, R. Verma, S.K. Lee, and E.R. Melhem. Diffusion-tensor mr imaging and tractography: Exploring brain microstructure and connectivity. *Radiology*, 245(2):367–384, 2007.

[PWS11] Nicholas Polys, Andrew Wood, and Patrick Shinpaugh. Cross-platform presentation of interactive volumetric imagery. Departmental Technical Report 1177, Virginia Tech, Advanced Research Computing, 2011.

[RSDH10] N. Ratnarajah, A. Simmons, O. Davydov, and A. Hojjat. A novel white matter fibre tracking algorithm using probabilistic tractography and average curves. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010*, pages 666–673, 2010.

[SAO10] S. Settapat, T. Achalakul, and M. Ohkura. Web-based 3d visualization and interaction of medical data using web3d. In *SICE Annual Conference 2010, Proceedings of*, pages 2986–2991. IEEE, 2010.

[Sch05] Henning Scharsach. Advanced gpu raycasting. *Proceedings of CESCG*, 5:67–76, 2005.

[SHC09] Mikhail Smelyanskiy, David Holmes, Jatin Chhugani, Alan Larson, Douglas M. Carmean, Dennis Hanson, Pradeep Dubey, Kurt Augustine, Daehyun Kim, Alan Kyker, Victor W. Lee, Anthony D. Nguyen, Larry Seiler, and Richard Robb. Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures. *IEEE Transactions on Visualization and Computer Graphics*, 15:1563–1570, November 2009.

[SKR10] Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. Xml3d: interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 175–184, New York, NY, USA, 2010. ACM.

[SMF00] Marcelo Rodrigo Maciel Silva, Isabel Harb Manssour, and Carla Maria Dal Sasso Freitas. Optimizing combined volume and surface data ray casting. In *WSCG*, 2000.

A Survey on Methods for Omnidirectional Shadow Rendering

Jan Navrátil
inavrati@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Jozef Kobrtek
xkobrt00@stud.fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Pavel Zemčík
zemcik@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

ABSTRACT

This paper focuses on methods of rendering shadows cast by point light sources. The goal is to summarize advantages and disadvantages of methods based on shadow mapping. We compare the traditional approach that exploits cube maps with the Dual–Paraboloid mapping. All of the methods are implemented on the latest hardware and they exploit capabilities of current GPUs. We also implemented optimization techniques which decrease the computational time. We examine the time the methods spent in particular rendering passes and we evaluate their overall performance. Finally, we conclude the comparison with some recommendations for typical applications in which the methods of interest can be exploited. We also suggest some direction of future investigation.

Keywords: shadow mapping, rendering, GPU, performance, cube maps, Dual–Paraboloid mapping

1 INTRODUCTION

Shadows play very important role in modern graphics applications as they increase overall visual cue from a rendered image. The shadow mapping algorithm [Wil78] and the technique based on shadow volumes [Cro77] are the most popular techniques for adding shadows to 3D scenes.

A well known disadvantage of the shadow mapping algorithm is the limited resolution of textures which store the depth information. Furthermore, it is also difficult to render shadows cast from point light sources. The basic shadow mapping algorithm cannot cover the whole environment with a single texture and thus additional computations are required. Such additional computations decrease the performance especially in scenes with a complex geometry.

The technique based on shadow volumes can easily render shadows from point light sources with per pixel accuracy. However, a high fill rate rapidly reduces the computational performance even for moderate sized scenes. Even though some optimization approaches exist [LWGM04], interactive applications mostly use the shadow mapping algorithm.

In this paper, we investigate several approaches for rendering shadows cast from point light sources based on the shadow mapping algorithm. Our contribution is the evaluation of the advantages and disadvantages of the approaches. We compare the existing methods especially with respect to their performance. We present some figures related to the time spent on generation of shadow maps on GPUs [MGR+05, Gru07] and also some frame times related to a camera view. We will also discuss the efficiency of all of the presented methods and potential implementation problems related to GPUs. Since the paper is restricted to the specific case of the shadow mapping algoritm we do not consider the shadow volumes approaches [LWGM04, VBGP09] as well as techniques that increase visual quality of shadows [WSP04]. Because they add some additional processing steps that might influence the results.

In Section 2, we refer to some techniques related to shadow rendering. We also mention some existing surveys. Section 3 introduces some issues that may arise when implementing the presented methods. We demonstrate all of the methods and their optimization in Section 4 and in Section 5, we present our experiments and discuss their results. We conclude our work in Section 6 where we also suggest some areas of future investigation.

2 RELATED WORK

For high quality shadow rendering, techniques such as ray tracing can be used. However, the shadow volumes algorithm or the shadow mapping approach are the most frequently used in interactive applications. The shadow volume technique [Cro77] provides per-pixel accuracy, its main disadvantage is a huge required fill rate. This fact does not allow for its common use in interactive applications. We can, however, find some methods that reduce the fill rate [LWGM04]. Nevertheless, the shadow mapping is the most popular algorithm for shadow ren-

dering. Basically, two approaches exist to render shadows cast from omnidirectional light sources using the shadow mapping algorithm. Firstly, shadow maps can be represented by faces of a cube map [Ger04]. In this case, six render passes are needed to fill the data into the cube map faces. Secondly, the Dual–Paraboloid mapping technique [BAS02, OBM06] can be used. It is capable of capturing the whole environment in two render passes. However, the mapping is not linear and thus not fully supported by contemporary graphics hardware. Recently, different techniques have been introduced [CVM11, HWL+11] that discuss other types of parametrizatons.

All of the above mentioned methods are capable of rendering shadows cast from omnidirectional (point) light sources. However, they all have some limitations and their usage may depend on the application and on scene complexity. Some surveys of shadow rendering have already been published, but they generally compare visual quality of the shadows with respect to the aliasing error [SWP10] or they address problem of soft shadow rendering [HLHS03]. In these cases, mostly directional light sources have been taken into account. The omnidirectional light sources need an extra treatment for creating shadow maps but also for reducing the aliasing error. Vanek et al. [VNHZ11] did some experiments with Dual–Paraboloid mapping technique but they did not work with the cube maps approach at all. They considered the cube map approach ineffective for omnidirectional light sources.

3 ISSUES OF THE SHADOW MAPPING ALGORITHM

3.1 Overview of the Algorithm

The first step of the shadow mapping algorithm is creation of the shadow map. A virtual camera is placed in the position of a light source. Then the scene is rendered as viewed from the virtual camera and the depth information is captured in the shadow map. In the second step, the scene is rendered from a camera point of view and the rendered pixels are compared with values stored in the shadow map.

During the process of the creation of the shadow map, the geometry has to be transformed to the light space coordinate system. For this purpose, the transformation matrix has to provide an appropriate transformation based on the type of the light source.

3.2 Linear Transformation and Interpolation

In case of directional light sources, orthogonal projection is used since all of the light rays have the same direction. For spotlights, perspective projection is used since the spotlights cover only certain part of the scene.

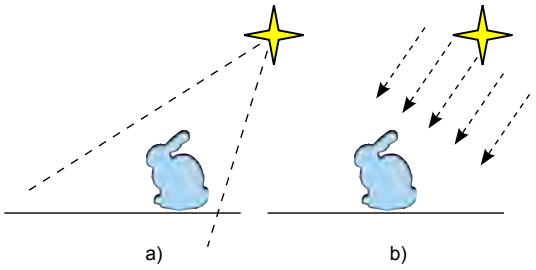


Figure 1: (left)A virtual camera for spotlights creates the view frustum. The frustum covers only a part of the scene based on a direction of the spotlight. (right) Directional lights use orthographic projection, because direction the light rays are parallel.

Then, the field-of-view in perspective projection is similar to the concept of falloff angle in spotlights. (see Figure 1). The perspective projection has a limited field-of-view range and thus it can not cover the whole environment. However, both projections are linear and thus they do not allow for covering the 180 degree field-of-view appropriately. To cover the whole environment, multiple linear projections are required. This means that if we want to use the basic shadow mapping algorithm for omnidirectional light sources, multiple render passes are necessary to create the shadow map (see Figure 2). Otherwise, a non-linear transformation has to be used.

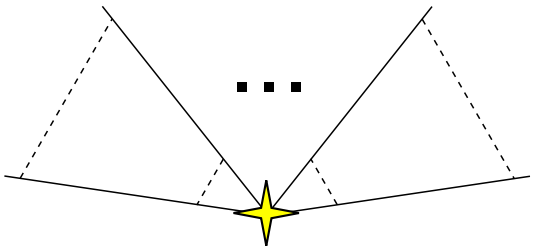


Figure 2: Multiple frusta have to be placed next to each other to cover the whole environment.

When we apply a projection, represented by a matrix, on vertices in the vertex shader, the fragments in the fragment shader are linearly interpolated. Instead of multiple linear projections, we can apply a non-linear transformation. The non-linear transformation, however, does not work well with the interpolation scheme used in graphics hardware. Straight lines are curved after the transformation (see Figure 3). It causes unwanted artifact for large polygons. The solution for these artifacts is to refine tessellation of the scene. For small polygons, the artifacts are not noticeable.

3.3 Limited Resolution

Stamminger et al. [SD02] described two types of aliasing: *perspective* and *projection*. Perspective aliasing is caused by limited resolution of shadow texture when

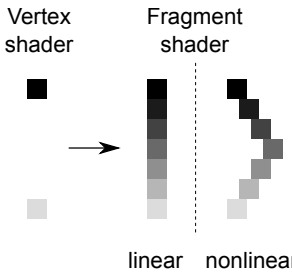


Figure 3: Fragments, that have to be rasterized between two vertices, are linearly interpolated in fragment shaders. Nonlinear parameterization can cause that fragments do not lie on a line.

the shadow map is undersampled while projection aliasing appears when the direction of light rays is parallel to the surface. Some methods exist that try to reduce the perspective aliasing artifacts on shadow boundaries. The shadow map can be filtered to make the shadow smooth [Fer05].

4 OVERVIEW OF METHODS

In this section, we present an overview of various methods for rendering shadows cast from omnidirectional light sources. We describe principles of each of the methods and we discuss their advantages and disadvantages. Furthermore, we present some optimization techniques that eliminate some of the disadvantages in order to achieve the best results for each of the methods. For our purpose, we only deal with omnidirectional light sources. It means that the light is emitted from a single point in space: therefore, we neglect an extent of the light source.

4.1 Cube Shadow Maps Technique

In Section 3, we mentioned how problematic it is to cover the whole environment with traditional projection transformations. In order to create shadow maps for an omnidirectional light source, it is necessary to point the virtual camera into six directions. The view direction of the virtual camera should point toward directions defined by the axes of the local coordinate system of the cube: positive X, negative X, positive Y, negative Y, positive Z and negative Z. This is almost identical to the way how a cube map for environment mapping is generated except that in this case depth values are stored instead of color.

Basics of the Cube Shadow Maps

The faces of the cube represent shadow maps and directions of the faces shows the particular direction for the virtual camera (see Figure 4). In order to cover the whole environment, the traditional shadow mapping algorithm exploits cube maps to visualize shadows cast from point lights. To fill the data in the cube shadow map, six render passes have to be performed. The GPUs generally support the cube shadow maps which are thus easy to implement.

The biggest disadvantage of the cube shadow maps is that six render passes are often too expensive. This fact can cause rapid decrease of performance for complex scenes with high number of polygons. Even if per-object frustum culling is applied, rendering of shadows is still very expensive compared to rendering of the rest of the scene.

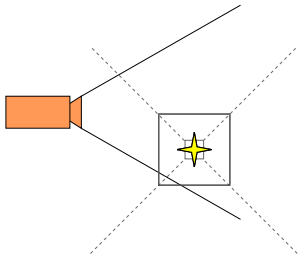


Figure 4: Illustration of the cube shadow maps technique. Each face of the cube stores depth values for a certain part of the scene.

Efficient Cube Face Frustum Culling

The methods of reduction of the number of passes have been investigated [KN05]. If the light source is outside the view frustum, then we can skip rendering of at least one face of the shadow map. This leads to the most noticeable effect on the performance

For our experiments, we use the following technique for efficient culling of cube map faces. A camera view frustum and each cube map frustum are tested for their mutual intersection. Those frusta that do not intersect can be discarded for further rendering because they do not affect the final image. The efficient culling of arbitrary frustum F against the camera view frustum V works as follows. The frusta are defined by 8 boundary points and 12 boundary edges. To determine whether the two frusta intersect, two symmetric tests have to be performed. Firstly, it should be tested whether a boundary point of one frustum lies inside other frustum (see Figure 5a). Secondly, it should be tested whether a boundary edge of one frustum intersects one or more clip planes of other frustum (see Figure 5b) [KN05].

For each face of the cube shadow map, we investigate whether the camera view frustum intersects the shadow face frustum and vice versa. If it is not the case, the shadow face frustum does not affect the scene and we can skip the additional processing (see Figure 6).

It is also necessary to take into account shadow casters outside the view frustum. If we cull the shadow caster against the view frustum, the projected shadow may still be visible in the view frustum. On the other

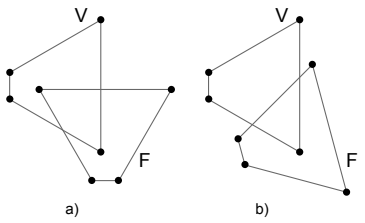


Figure 5: A frustum consists of boundary points and boundary edges. Two frusta intersect when (a) at least one boundary point of the frustum F lies inside other the frustum V or (b) at least one boundary edge of the frustum F intersects a face of the frustum V.

hand, culling the shadow caster against the cube map frustum draws invisible shadows as well. King et al. [KN05] suggest to use frustum-frustum intersection test described above for the shadow casters as well. Since we use point light sources, rays are emitted from a single point towards all shadow casters. This is analogous to the perspective projections. If the shadow casters are enclosed by bounding objects, frusta representing the projected shadows can be created [KN05] and then the frustum-frustum test can be applied in this case as well. These tests are performed once per frame.

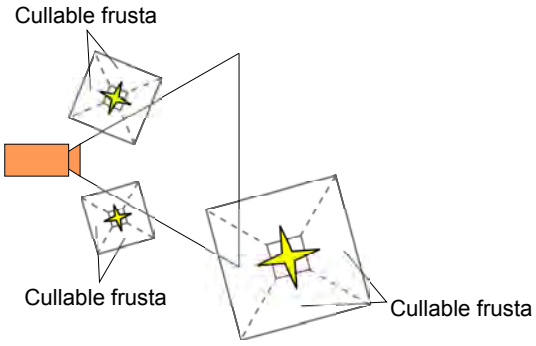


Figure 6: If the light source lies outside the camera view frustum, at least one face is cullable.

4.2 Dual–Paraboloid Mapping Algorithm

In the following text, we will discuss the Dual–Paraboloid Mapping algorithm (DPSM) [BAS02]. The mapping is based on two paraboloids attached back-to-back, each capturing one hemisphere:

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2+y^2), \quad x^2+y^2 \leq 1 \qquad (1)$$

In principle, a single hemisphere mapping can be imagined as an exploitation of a totally reflective mirror which reflects incident rays from the hemisphere into the direction of the paraboloid (see Figure 7). The rays may carry some information about the environment (mostly distance to the light) and the information can be stored into a texture. The texture coordinates are computed according to coordinates of the point where the ray is reflected. The Dual–Paraboloid mapping basically maps 3D space to 2D which is represented by the shadow map.

The algorithm needs only two render passes to capture the whole environment. Thus, it is more efficient than the cube shadow maps technique. Other parameterization can be certainly found (spherical, cube mapping etc.) but the proposed parabolic parameterization maintains its simplicity and performance, e.g. in GPU implementation [OBM06]. It minimizes the amount of used memory and the number of render passes that are necessary to cover the whole environment.

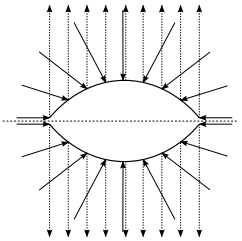


Figure 7: Two paraboloids attached back-to-back can capture the environment from all directions.

Nevertheless, the DPSM algorithm has also some disadvantages. While in the cube shadow map approach, all the transformations needed to create the shadow map are linear, they do not need any extra treatment on GPUs. This mainly concerns interpolation process between vertex and fragment shader (see Sec 3). When using the DPSM algorithm, the rendered scene needs to be finely tessellated because the mapping is not linear and thus it does not work well for large polygons. It may, however, introduce new bottlenecks.

4.3 Limitations of Geometry Shader

It is also possible to exploit both shadow mapping methods utilizing a geometry shader in order to reduce the number of render passes from six (two in Dual–Paraboloid mapping algorithm) to a single one [Eng08]. In this case, we exploited capablities of the frequently used graphics card, i.e., NVIDIA GeForce GTX560Ti, which supports geometry shaders.

The core of this method is usage of multiple render targets for and rendering all of the six cube map faces at once. The geometry shader transforms each of the incoming triangles with view-projection matrix of the corresponding cube. A naive approach sends all the incoming geometry data to all render targets, producing three to five times more geometry data than necessary. Such data is, however, anyhow discarded in the following rendering phases by the rasterizer. This leads to a massive performance penalty, as seen in Table 1. The results were measured on the same scene with the shadow map resolution set to 1024^2.

	avg. FPS
Cube6	6.19
Cube6Optim	20.3
DP	18.81
DPOptim	30.90

Table 1: All the methods exploit geometry shader and render the shadow maps in one pass.

This method was further optimized by testing each object bounding sphere against view frusta of the cube map faces, or, in case of Dual–Paraboloid mapping algorithm, against plane dividing scene in place of both paraboloids. Cube shadow mapping method was sped up by 227%, but still resulting in a very poor performance. Dual–Paraboloid mapping approach did not benefit that much from optimization, resulting in only 64% increase of performance, but also scoring far less than multi-pass methods.

Despite the optimizations, these methods did not overcome above mentioned optimized 6-pass techniques (described in Section 4.1). The core problem of the geometry shader is its execution model. It outputs data in a serial fashion with no parallelism used. Utilizing vertex shader and multiple passes overcomes the above mentioned geometry shader solutions despite switching of the render targets and updating resources between render calls.

5 EXPERIMENTAL RESULTS

We implemented the experimental framework in DirectX11 on an Intel Core i5 CPU 661 running at 3.33GHz using NVIDIA GeForce GTX560Ti GPU. The rendered images have resolution of 1024×768. We used the 32bit render target for the shadow maps. The resulting graphs were generated from an experimental walkthrough of a demo scene. The benchmarking scene had approximately 3 millions of vertices.

Our implementation does not introduce any hardware specific features. We can assume that the difference between the approaches would not be principally different.

5.1 Frame Time in Walkthrough

The first measurement shows dependence of the frame time for the walkthrough of the scene for all of the implemented methods. The unoptimized variants of the cube shadow maps and the Dual–Paraboloid shadow mapping (DPSM) show the worst results. In this approach, for every pass, all the geometry is rendered. Naturally, six render passes of the cube shadow maps lead into the highest frame time.

The basic optimization technique provided the bounding object frustum culling against the view frustum, the cube shadow maps frustum and the clipping plane for paraboloids. In this case, the same amount of geometry is rendered in both approaches. The overhead for increased number of the render passes for the cube shadow maps had no effect on an overall time for a single frame and thus the resulting frame times are similar.

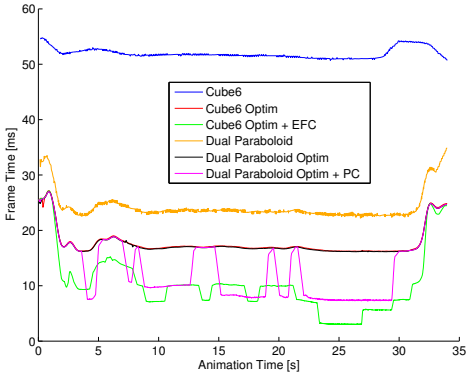


Figure 8: Frame times for the walkthrough of the scene for all implemented methods.

The cube shadow maps approach exhibits the best result with the effective cube face frustum culling - *EFC* (see Section 4.1). The plot shown in Figure 8 shows that the DPSM increased the performance only by skipping one paraboloid wherever appropriate (using plane clipping - *PC*). Otherwise, all of the geometry had to be rendered in two passes. The cube shadow maps approach can skip up to five render passes and thus it achieved the best results (e.g. in 25th second of the walkthrough). The frame time in the DPSM depends mainly on the amount of rendered geometry and also the amount of geometry in the given hemisphere. As can be seen in the plot, the DPSM saved only 50% of the computation time when rendering the scene only for one side. However, the cube shadow maps saved up to 83% of the performance. Furthermore, Figure 9 shows that the DPSM uses only one paraboloid most of the time and also that the cube shadow map rarely performed all six passes. This is mainly because the light source lied outside the camera view frustum.

5.2 Timings of Render Passes

Since the shadow mapping algorithm renders shadows in two passes, we investigated frame times for the passes for all implemented methods. The time for final shadow rendering showed to be equivalent for all methods, because it mainly depends on number of rendered geometry. Here, the view frustum culling was employed. The most noticeable differences were in times for generation of the shadow map.

As shown in Figure 10, the methods without any optimization had to render all the geometry six times in case of the cube shadow maps (blue) or two times in case of

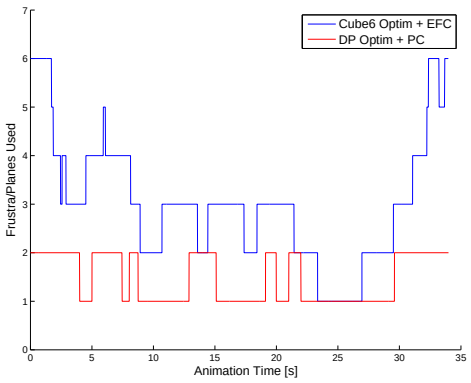


Figure 9: The plot shows the number of processed cube faces (blue) and the number of rendered paraboloid sides (red).

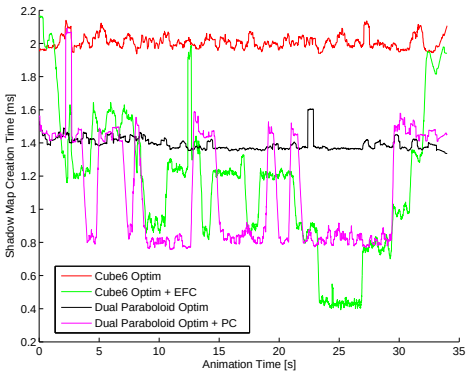


Figure 10: Evaluation of the times which all methods spent on the shadow map generation. For better illustration, unoptimized methods are not visible, because they had very poor results as compared to optimized techniques.

the DPSM algorithm (yellow). There are also some differences between methods where a frustum and plane culling is applied. The DPSM algorithm was faster compared to the cube shadow maps. An overall amount of rendered geometry was equivalent in both cases so there seems to be some additional overhead in the cube shadow maps technique.

Generally, the DPSM algorithm was faster when only one paraboloid was processed. The cube shadow map technique reached the similar times when only 2 faces were processed. The plot in Figure 10 also shows that in time 25 s, the cube shadow maps technique achieved the best results. In this case, only one face was processed which is mainly based on the position of a light sources relative to a camera (see Figure 11).

5.3 Effect of Shadow Map Resolution

We also investigated how the shadow map resolution affects the frame rate. In Table 2 and Table 3 you can see the results for various shadow map sizes. As you can see, the optimization techniques brought an increase in frame rate.

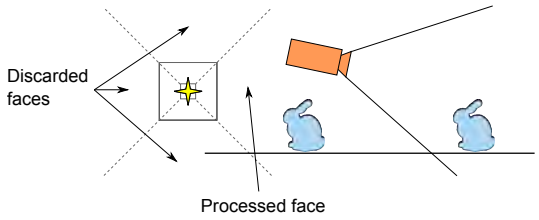


Figure 11: An illustration of the situation when only one face is processed during shadow map generation pass. Figure shows that only one cube face frustum intersects with the camera view frustum.

Considering shadow map as a texture storing single 32-bit value per texel, memory consumption of the cube shadow maps was from 24MB (1024×1024) to 384MB (4096×4096). Whereas the Dual–Paraboloid mapping approach uses one third of memory compared to the cube shadow maps (8MB to 128MB), it is more computationally intensive. Utilizing efficient frustum culling methods, we can save computation time by reducing number of the render passes and size of the geometry data, which also reduces memory utilization (less number of values stored due to frustum culling).

When taking 1024^2 resolution of shadow map as 100% performance for each method, switching to 2048^2 causes performance drop off only by 6.54% in average, but greatly increases shadow quality. Choosing 4096^2 resolution for shadow map takes 25.76% performance penalty in average.

Image quality of the result of Dual–Paraboloid mapping technique depends on the geometry of the occluding object. As described in [BAS02, OBM06], the Dual–Paraboloid mapping causes low-polygonal casters to produce incorrect shadows. Increasing shadow map resolution does improve shadow quality, but still can not match the quality of details achieved by the cube shadow maps approach (see Figure 12).

	1024^2	2048^2	4096^2
Cube6	75.71	70.04	47.9
Cube6Optim	150.43	116.76	64.04
Cube6Opt+EFC	188.71	151.67	89.68
DP	167.95	146.62	97.52
DPOptim	207.24	178.67	109.4
DPOptim+PC	208.15	180.24	110.95

Table 2: FPS of low-poly scene (600K vertices)

5.4 Position of a Light Source Relative to Geometry

We also performed an experiment where we focused on position of a light source relative to the geometry. This experiment was inspired by techniques for computation of interactive global illumination [RGK+08].

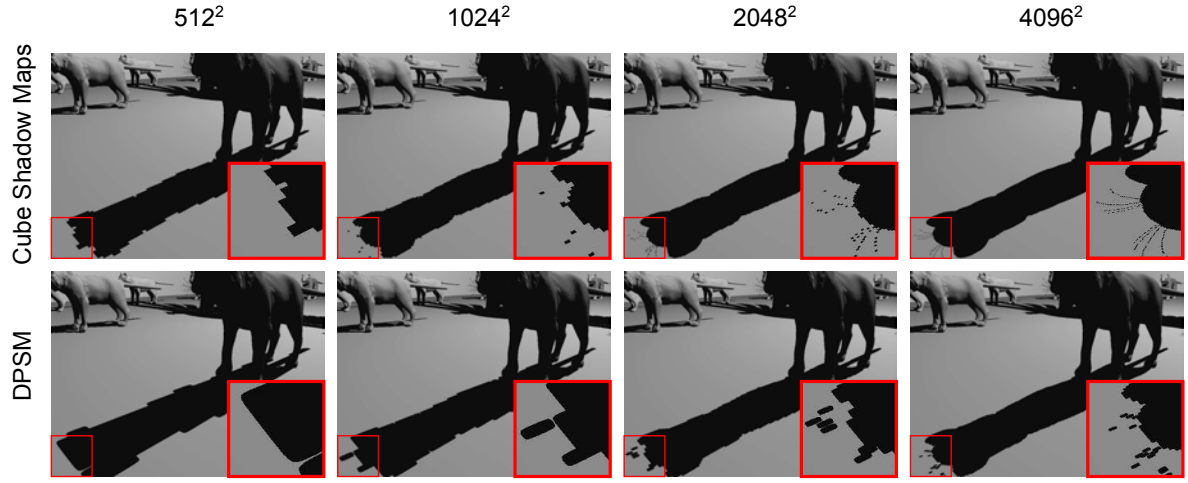


Figure 12: Figure shows how the shadow map resolution influences the shadow quality. Since a single paraboloid covers one hemisphere, one shadow map texel is projected on the large area in the scene (as compared to the cube shadow maps). This leads to worse quality of shadows.

	1024^2	2048^2	4096^2
Cube6	19.11	18.38	16.21
Cube6Optim	57.15	51.23	36.50
Cube6Opt+EFC	127.47	114.21	83.38
DP	41.50	39.74	33.17
DPOptim	57.47	54.32	42.85
DPOptim+PC	90.56	86.08	69.58

Table 3: FPS of high-poly scene (3M vertices)

In this case, *Virtual Point Lights* (VPLs) are generated on the surface to approximate indirect lighting. The reflected light is scattered into all directions. Therefore, some method is required to handle shadows from the reflected light. For this purpose, all the geometry data is positioned into one hemisphere relative to the light source. When the geometry is distributed around the light sources, it is useful to use the cube shadow maps technique, because it has better optimization strategy and it can easily manage the number of processed cube map faces. However, when we need to render only one hemisphere, the DPSM algorithm is more sufficient.

We measured times for generation of the shadow map in both of the presented techniques. Ritschel et al. [RGK+08] employed the Dual–Paraboloid mapping algorithm in their approach. They generated shadow maps for multiple VPLs (256 and more) from simplified geometry. We compared timings for the DPSM and the cube map technique.

In Figure 13, it can be seen that the DPSM algorithm is approximately two times faster than the cube shadow maps approach. The results are similar for various levels of the scene complexity. The Dual–Paraboloid mapping algorithm can be used despite its worse accuracy, because indirect lighting produces low-frequency shadows. In this case, the artifacts are blurred.

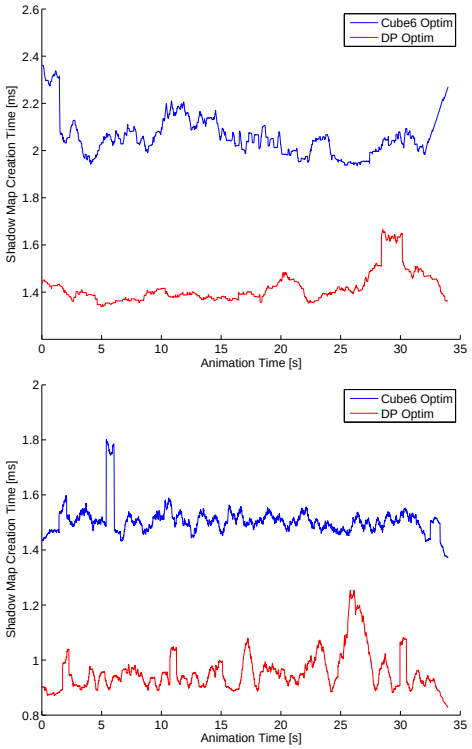


Figure 13: Figure illustrates times that the methods of interest spent on generation of the shadow map. In this case, the geometry is placed into one direction from the light source. The scene was represented by points only: 3 millions points (Top) and 100k points (Bottom).

6 CONCLUSION AND DISCUSSION

The goal of the work presented in this paper was to investigate the shadow mapping algorithm and techniques based on this algorithm as well as their capabilities to render shadows cast from point light sources. We ex-

amined two techniques that are based on the shadow mapping algorithm. The cube shadow maps approach exploits the traditional shadow mapping algorithm and renders the shadow map on cube faces. The Dual–Paraboloid shadow mapping uses nonlinear parameterization to render one hemisphere in one render pass.

The initial assumption was that multiple render passes performed by the cube shadow maps technique should be very time consuming process. The result of the measurement is that an unoptimized version of the cube shadow maps exhibits the worst performance of the examined algorithms. When a simple optimization technique was used significantly increased performance was reached, in fact, the best of the examined algorithms. The performance and the visual quality of the cube shadow maps is better compared to the Dual–Paraboloid algorithm. However, the Dual–Paraboloid algorithm produces better results if we consider the specific position of a light source related to a geometry, e.g., when computing illumination using VPLs.

Future work includes the complexity study that will improve the quality of measurements but since the timings depend mainly on the rendered geometry, however, as the complexity class is similar for all approaches, no significant differences are expected. It might be interesting to compare the implementation using the current hardware capabilities, e.g. CUDA. Evaluation of visual quality of the presented methods and their ability to deal with the aliasing problem in the shadow mapping algorithm is also subject of future work.

ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070 and the Artemis JU project R3-COP, grant no. 100233.

REFERENCES

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *J. Graph. Tools*, 5(1):9–22, January 2000.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Proceedings of Computer Graphics International*, pages 397–408, 2002.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, 1977.

[CVM11] Marcel Stockli Contreras, Alberto José Ramírez Valadez, and Alejandro Jiménez Martínez. Dual sphere-unfolding method for single pass omni-directional shadow mapping. In *ACM SIGGRAPH 2011 Posters*, SIGGRAPH '11, pages 69:1–69:1, New York, NY, USA, 2011. ACM.

[Eng08] Wolfgang Engel, editor. *Programming Vertex, Geometry, and Pixel Shaders*. Charles River Media; 2 edition, 2008.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[Ger04] Philipp Gerasimov. Omnidirectional shadow mapping. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, pages 193–203. Addison Wesley, 2004.

[Gru07] Holger Gruen. Performance profiling with amd gpu tools: A case study. AMD Sponsored Session, GDC, March 2007.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[HWL+11] Tze-Yiu Ho, Liang Wan, Chi-Sing Leung, Ping-Man Lam, and Tien-Tsin Wong. Unicube for dynamic environment mapping. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):51–63, January 2011.

[KN05] Gary King and William Newhall. Efficient omnidirectional shadow maps. In Wolfgang Engle, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 435–448. Charles River Media, Hingham, MA, 2005.

[LWGM04] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. Cc shadow volumes. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 146–, New York, NY, USA, 2004. ACM.

[MGR+05] Victor Moya, Carlos Gonzalez, Jordi Roca, Agustin Fernandez, and Roger Espasa. Shader performance analysis on a modern gpu architecture. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 355–364, Washington, DC, USA, 2005. IEEE Computer Society.

[OBM06] Brian Osman, Mike Bukowski, and Chris McEvoy. Practical implementation of dual paraboloid shadow maps. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 103–106. ACM, 2006.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8. ACM, 2008.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562. ACM, 2002.

[SWP10] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. In *EUROGRAPHICS 2010 State of the Art Reports*, 2010.

[VBGP09] Forest Vincent, Loïc Barthe, Gael Guennebaud, and Mathias Paulin. Soft Textured Shadow Volume. *Computer Graphics Forum*, 28(4):1111–1120, 2009.

[VNHZ11] Juraj Vanek, Jan Navrátil, Adam Herout, and Pavel Zemčík. High-quality shadows with improved paraboloid mapping. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part I*, ISVC'11, pages 421–430, Berlin, Heidelberg, 2011. Springer-Verlag.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.

[WSP04] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. *In the Eurographics Symposium on Rendering*, 2004.

TimeSeriesPaths: Projection-Based Explorative Analysis of Multivarate Time Series Data

Jürgen Bernard
Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
juergen.bernard@igd.fraunhofer.de

Nils Wilhelm
Technische Universität Darmstadt, Germany
nwilhelm@rbg.informatik.tu-darmstadt.de

Maximilian Scherer
Interactive Graphics Systems Group, Technische Universität Darmstadt, Germany
maximilian.scherer@gris.tu-darmstadt.de

Thorsten May
Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
thorsten.may@igd.fraunhofer.de

Tobias Schreck
Data Analysis and Visualization Group, Universität Konstanz, Germany
tobias.schreck@uni-konstanz.de

ABSTRACT

The analysis of time-dependent data is an important problem in many application domains, and interactive visualization of time-series data can help in understanding patterns in large time series data. Many effective approaches already exist for visual analysis of *univariate* time series supporting tasks such as assessment of data quality, detection of outliers, or identification of periodically or frequently occurring patterns. However, much fewer approaches exist which support *multivariate* time series. The existence of multiple values per time stamp makes the analysis task per se harder, and existing visualization techniques often do not scale well.

We introduce an approach for visual analysis of large multivariate time-dependent data, based on the idea of projecting multivariate measurements to a 2D display, visualizing the time dimension by trajectories. We use visual data aggregation metaphors based on grouping of similar data elements to scale with multivariate time series. Aggregation procedures can either be based on statistical properties of the data or on data clustering routines. Appropriately defined user controls allow to navigate and explore the data and interactively steer the parameters of the data aggregation to enhance data analysis. We present an implementation of our approach and apply it on a comprehensive data set from the field of earth observation, demonstrating the applicability and usefulness of our approach.

Keywords: Multivariate Time Series, Visual Cluster Analysis, Exploratory Data Analysis, Data Projection, Data Aggregation

1 INTRODUCTION

Multivariate time series data are gathered in many domains including economics, experimental physics, computer vision, robotics, and earth observation. E.g., in the financial domain, large amounts of stock prices are tracked over time; in earth observation, daily temperatures and many additional parameters are observed at specific locations over time; time-dependent measurements also arise in monitoring traffic parameters on a communication network. Analysis of time series data can take many forms, including assumption-free exploration; correlation of time series with each other; or evaluation of specific generative models. Much work has been done focused on analyzing one-dimensional time series, and respective solutions are often applied to multivariate data by analyzing each dependent variable versus an independent one. However, for multivariate data the widely used *IID* assumption (independent and identically distributed) usually does not hold. Therefore there is a need to analyze all dimensions of such data at once.

In the context of data mining and visual analytics, multivariate time series analysis is a difficult problem, with solutions typically relying, in some form or the other, on dimensionality reduction, feature selection, projection, and glyph-based visualization. The task at hand often includes finding periodic or frequent patterns in

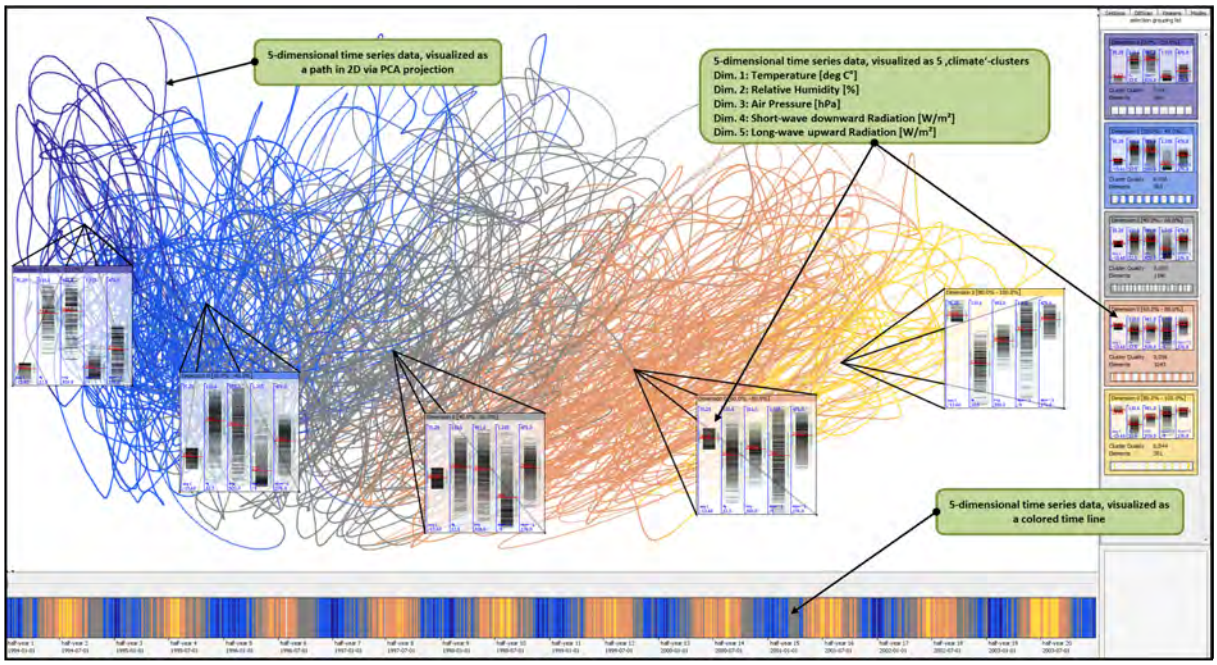


Figure 1: Main display: data analysis of a multivariate time series of 10 years length is always challenging due to overview problems and visual cluttering. This is the starting point of our data exploration. The Time Series Path System provides visual structures and interactive functionality to address the implied challenges. In this example, we aggregate a weather scenario by its temperature values and receive 5 well-distributed data clusters from cold (blue) on the left to warm (yellow) on the right. This is a qualified starting point for selection and filtering approaches to detect periodicity, dense data regions and outliers. Confer our case study in Section 5 for details about the 2D projection.

the data, relating multiple variables to each other, or detecting outliers or anomalies. Visual-interactive approaches can help to tackle these challenging tasks by closely involving the user in the exploration process, addressing the typically difficult parameter selection problem, which could be more complicated to solve relying on purely automatic methods.

Several works propose to visually analyze multivariate time-dependent data by dimensionality reduction [26, 12]. Multivariate data is visualized as two-dimensional time series paths obtained by dimensionality reduction (projection to 2D). While these works visually compare sections of labeled multivariate time-dependent data, they do not consider exploratory search in unknown data sets. Furthermore, these works do not focus on aggregation efforts to reduce over-plotting problems. To this end, we introduce interactively steerable data aggregation, supporting handling of multivariate time series data. In particular, the user is able to group data points according to data-specific characteristics like statistical calculations based on value and time, or clustering results.

Our approach supports an effective overview of frequent and infrequent *states* in multivariate time series data even in cases of very large data. Furthermore, users can interactively select meaningful path line subsets for detailed exploration and for visual clutter reduction purposes. Understanding of aggregated data groups is supported by showing a comprehensive cluster glyph metaphor, wherever data aggregation visualization is required within the exploration process. We directly involve the user in the exploration process, combining data exploration with interactive steering of the automatic analysis methods, such as searching for appropriate clustering parameters, in particular.

We demonstrate the usefulness of our approach by an application to earth observation data. There, long time series of many parameters arise, and users want to understand periodicities, trends, and anomalies. We show how our set of interactive views allows for interactively exploring weather patterns of different lengths and parameters. Due to our data aggregations, domain users can explore multivariate weather data in a single display, giving an overview of all data aspects at once.

The remainder of this paper is structured as follows. In Section 2 we discuss related work in several areas. In Section 3 and 4 we motivate our approach, explain our system design and describe user interaction techniques. In Section 5 we apply our implementation to a real-world data set, demonstrating the usefulness of the approach. Finally, we summarize this paper and discuss future extensions in Sections and 6 and 7.

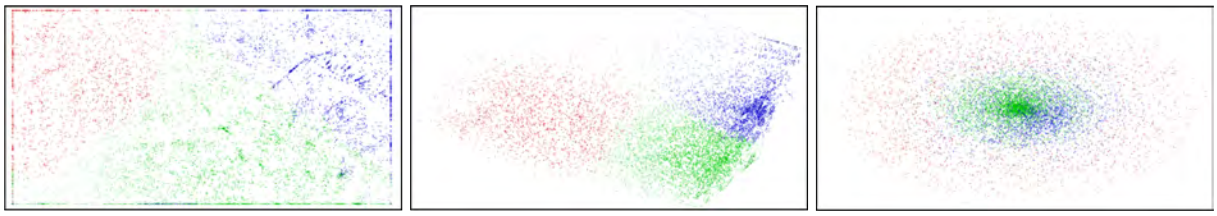


Figure 2: Visual comparison SOM, PCA and MDS projection technique. A k-means clustering result is shown.

2 RELATED WORK

Our work is related to analysis methods for time-dependent data and multivariate data. Time series analysis in general is conducted to understand the behavior of systems, to distinguish regular from extraordinary characteristics [14] and to predict future development [13].

Visualization of Time Series Data

The visualization of time series is often helpful for exploratory analysis. Traditionally, time series can be visualized by line charts [24]. However, using line charts is typically not effective for large time series data, as many and long time series lead to over-plotting if packed into a given display or would require excessive user navigation (cf. the problem specification in Figure 1). The *pixel paradigm* [2] for visualization of time series suggests to map the quantitative values of a time series to an appropriate color scale. Ultimately, each value can be represented by a single pixel. The Recursive Pattern technique [2] employs the pixel paradigm to arrange time series in a generic proximity-preserving way, allowing to arrange between row-by-row up to more complex patterns following space-filling curves. The comparison of many time series can be supported by rendering them next to each other in an appropriate display.

Besides side-by-side schemes, e.g., TreeMap-like layouts have been proposed [10]. An alternative to the pixel paradigm is to map the time axis to a spiral, effectively using more length, which is particularly useful for analysis of periodic data [27]. For domain-specific visualization tasks, e.g., atomistic simulation data, specialized techniques have been proposed [6]. An overview of time series visualization can be found in the textbook by Aigner et al. [1]

Automatic Support

Automatic analysis techniques are often used in time series visualization. E.g., the size of the data to be visualized may be reduced by aggregation [5] or dimensionality reduction [8].

In [25] prototypical time series patterns are found by cluster analysis, and linked to occurrence on the time scale by color-coding. In [17] a discretization approach is applied to support visual analysis of frequent subsequences in a node-link-diagram. Often, the similarity definition between time series or subsequences thereof is important to support exploratory search. In [28] so-called Perception Points of Interest are identified to sort a large number of time series for effective overviewing. Various other systems support the interactive retrieval of time series by defining appropriate similarity notions and query interfaces [9, 11, 3]. A visual-interactive approach to analyzing different families of functions is presented in [16]. Here, the authors allow the user to highlight data patterns of interest and provide linked views of the multidimensional data and the user-selected highlights.

Multivariate Time Series

The above methods mainly consider univariate time series. Yet, multivariate time series analysis is of importance in many domains. A number of approaches include small multiple displays for showing all variables over time next to each other. They may rely on line charts, pixel displays, or any other appropriate base technique. Also, automatic analysis methods for exploratory analysis in multivariate time series have been considered. E.g., in [19] a frequent-pattern-based approach is used to find interesting time series patterns along several levels of abstraction.

Recently, a number of authors have considered the visualization of multivariate time series data based on projection. The basic idea is to project discrete points in time to a 2D display, which in turn allows for analysis of the time series for regularities and irregularities [23]. In [22, 12] multivariate observation measures from motion tracking are projected using the Self-Organizing Map (SOM) method [15]. Individual observations are connected by lines, and glyphs illustrating the particular configurations of the motion are shown. In [18] multivariate time series are extracted from text, by computation of certain text features for discrete intervals along the sequence of the text. A PCA-based display was used to assess the development of the text content, by analysis of feature trajectories observed in the display. In [26] the authors use PCA-based projection to explore the sequence of small fixed-size intervals (so-called *n-grams*) of long univariate time series data. The approach was applied to stock market data and shown to provide an informative overview over long time series

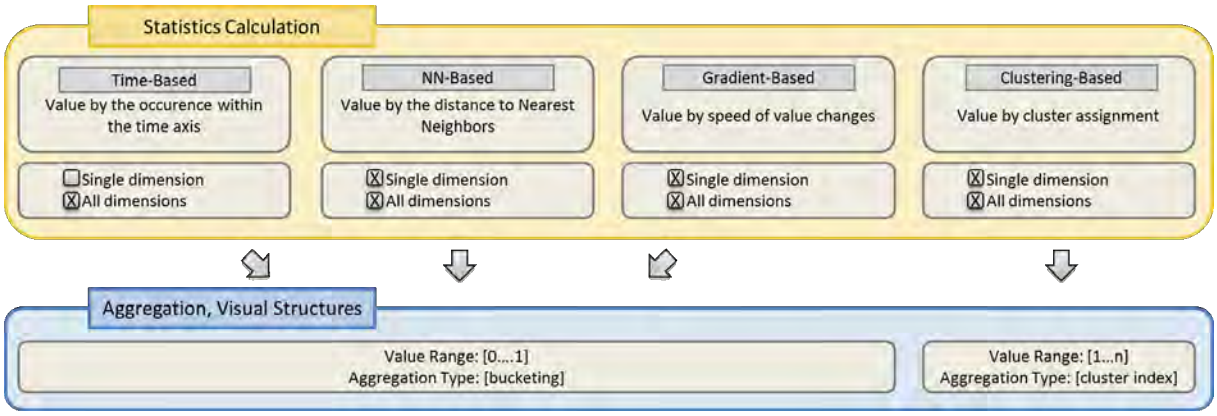


Figure 3: Aggregation of multivaritate time series data based on a variety of statistical data properties. Most of the functionality can either be performed on a single, or all dimensions.

data. In particular, the authors proposed their method to support the following analysis cases: Detection of cyclic behaviors; visual identification of base patterns and outliers; and analysis for trends.

3 SYSTEM DESIGN

In this work we present *TimeSeriesPaths*, a system for the analysis of multivariate time series data. The PCA projection algorithm is applied to arrange multivariate time-series on the (2D) display screen (the *Time Series Path Map*). We connect temporally adjacent data elements and receive a sequentially ordered set of points – a so called *time series path*. By default, such a visualization suffers from severe *over-plotting* and *overview problems*. In order to make such a visualization understandable for domain-experts and to counter the implied challenges, our approach comprises three contributions:

1. We apply semi-automatic data aggregation functionality, either derived from statistical data calculation, or from visual-interactive data clustering (cf. Subsection 3.2). This helps the user to get an *overview* to the dataset.

2. We present a cluster visualization technique that incorporates multiple information about the aggregated data (cf. Subsection 3.2). This supports *data interpretation and cluster comparison* approaches.

3. We propose a multi-view system with broad visual-interactive analysis functionality (cf. Subsection 4). Selection and highlighting modalities of data path subsets counter the challenge of *over-plotting* and allow for comprehensive *detail on demand* perspectives.

3.1 Visualizing Multivariate Time Series

Data Projection

We apply a projection technique to visualize multivariate time series data on 2D displays. An applicability consideration between visualizations based on projection and the multiple linechart technique is given in Section 6.

A general requirement concerning projection is the preservation of data topology, by means that similar data in the multivariate input space is also arranged close to each other in the display space. Due to their popularity and their diversity in arithmetical manner we chose PCA, SOM and Multidimensional Scaling MDS [7] as promising candidates. After an evaluation of our requirement criteria and a visual comparison in Figure 2, we choose the PCA algorithm as a default for prospective projection needs in the TimeSeriesPaths system. The non-linear MDS proves to be rather unsuitable for our approach, solely because it has troubles in separating k-means clusters. The SOM algorithm suffers in respect to the calculation speed and a major difficult (fully automatic) parametrization. Yet the key benefit of PCA derives from the ability to project data points in a linear manner, by means that the projection results do not lack on local distortions and thus allow for a more straight forwarded interpretation. Furthermore, the visual comparison of the three projection techniques shows a good cluster separation by PCA. We accept that PCA does not exploit the complete display space as well as the SOM projection. However later in this section, we will present our cluster glyph and show how our glyph visualization mitigates this problem.

Visualizing Time Series Paths

The visualization of time series paths is provided by the Time Series Path Map in the center of the display. Based on our data projection method, we connect individual data points by their chronological order to form paths. The projection arranges similar data points close to each other and reflects the data point distances of the multivariate data input space. Accordingly, if path sequences are similar to each other, their possibly close

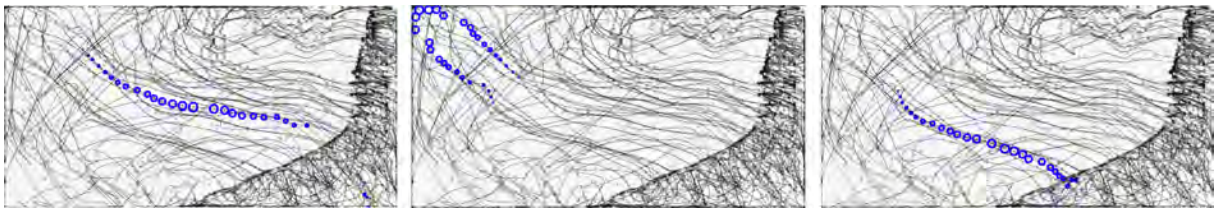


Figure 4: The "Rollercoaster Animation". By dragging the time slider, the user can explore the temporal development of the time series path. The cursor position and time series path neighbors are animated with circular shapes.

positions on the display space help the user with profound analysis approaches.

3.2 Multivariate Time Series Aggregation

Statistics Calculation and Aggregation

We integrate automatically generated statistical data information into the visualization to counter the overview problem and support the analysis process. So far, related approaches color-code data points for time-based and value-based changes or class labeling [12, 26]. Our approach generalizes this by a variety of statistical data measurements that provide additional important information, an overview is given in Figure 3. Altogether, our system provides four different properties of statistical data information for color coding:

- (a) occurrence within the time line (time-based)
- (b) nearest neighbor distance (NN-based)
- (c) speed of value change to adjacent time stamps (gradient-based)
- (d) cluster membership (clustering-based)

Except for (a), all statistical data information can either be calculated on a single dimension or on all dimensions of the data set. Thus, we are also able to perform domain-specific exploration tasks due to the level of detail in the aggregation setup. The number of data groups k can be specified by the user for all properties, (a)-(d).

Data color codings according to group affiliations are displayed on the Time Series Path Map, our time axis display at the bottom (called *Color Time Bar*), and the *Data Aggregation List* on the right, respectively. Showing multiple aspects of the data enables to find an appropriate aggregation level, to interpret groups of data and derive mutual characteristics, to detect outliers and to explore periodic behavior in the data.

In our case study (cf. Section 5), we will show that distributions of aggregated statistical data information on the Time Series Path Map and the Color Time Bar give valuable information about dense data regions, data anomalies and the periodicity of time series paths.

Generic Cluster Glyph

The aggregation of data into groups requires a meaningful cluster visualization method (cf. Figure 5). The main requirement is genericity in order to suit to a great variety of multivariate time series data. Additionally, averages, minima and maxima, variances, number of elements and cluster quality indices are needed. Each data dimension is displayed with an error bar chart glyph metaphor and labeled with the corresponding physical unit. Additionally, we include the distribution of time stamps on a time axis to monitor chronological data characteristics to detect periodic behavior or anomalies. Finally we demand the cluster glyph to show the cluster color coding for linking, and a headline for user-centered data labeling purposes.

Earlier we argued that PCA does not capitalize the entire border areas of the display space. We benefit from this instance due to the fact that we have free space remaining to position cluster glyphs for data aggregation operations. Four black concentrical lines connect the cluster glyph with the appropriate display coordinate without producing too much occlusion (see Figures 1, 6, 7 and 8).

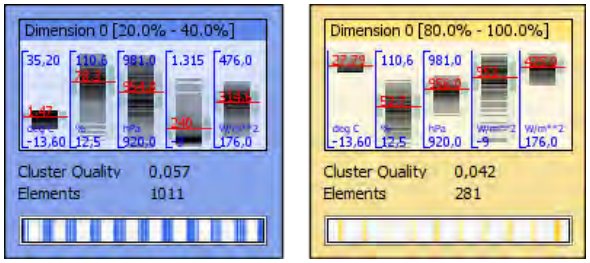


Figure 5: Generic Cluster Glyph. A boxplot-like visualization shows the distribution of data elements in each dimension of the dataset, transparency is used to show data frequency. Cluster centroid values are displayed as red bars, just like gray variance bands mapped laterally for each dimension. Statistical information about the data cluster is shown at the center, the data distribution on the global time axis is shown at the bottom.

4 INTERACTION TECHNIQUES

TimeSeriesPaths includes a set of linked user interaction modalities which work across the three different

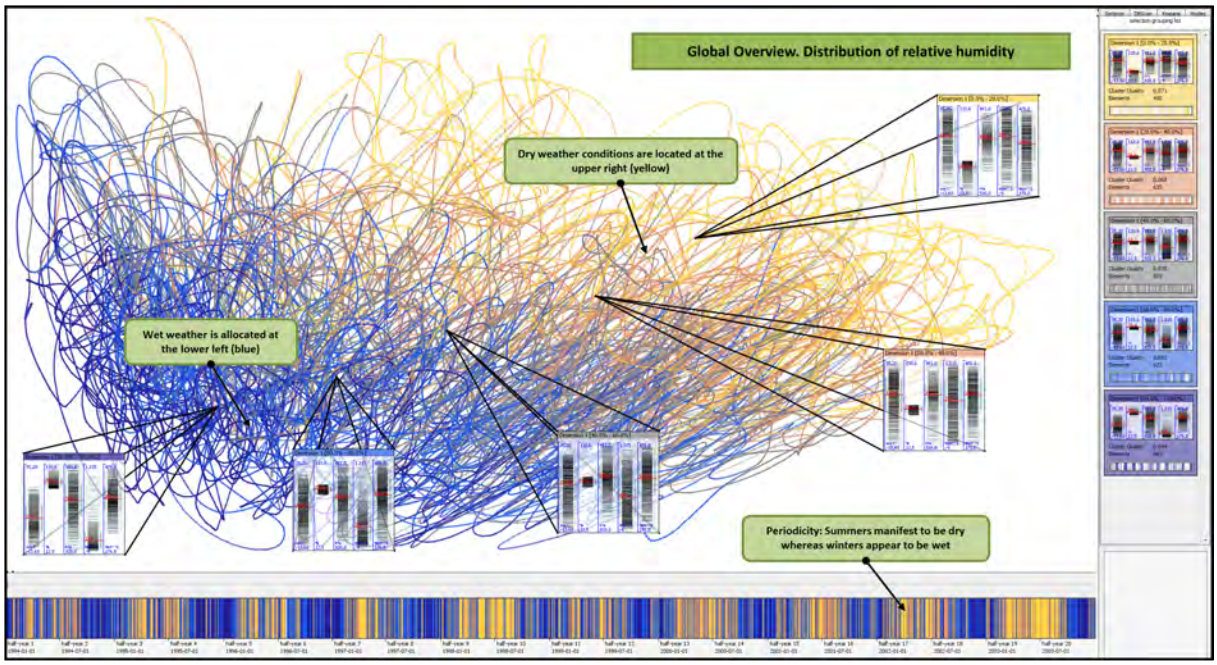


Figure 6: Data aggregation on single input data dimensions: Distribution of relative humidity values (color map: blue means wet, yellow means dry). We constitute rainy weather states to be located left on the Time Series Path Map. By exploring the Color Time Bar, we discover rainy weathers dominating the winter periods. Composing these two findings, we reason that (wet) winter climates are located on the left of the Time Series Path Map.

views. We give a short introduction to the major visual-interactive capabilities of the TimeSeriesPaths system.

Tooltipping

An important user requirement is detail on demand visualization. By hovering above data elements on the Time Series Path Map and the Color Time Bar, tooltips show the multivariate data information and the position of the respective data elements on the time axis (cf. Figures 9 and 10).

Selection, Interactive Grouping and Highlighting

The selection of data is supported in each of our three views. The user can (1) select single data points, (2) time series paths or subsequences thereof, (3) the selection of data within a distinct display region in the Time Series Path Map is possible (cf. Figure 7). The user sketches an arbitrarily polygonal shape on the map, and the surrounded data points will be selected.

Data selections can subsequently be added to the Data Aggregation List for additional information about the selection and for future re-selection. The respective selection is highlighted in all three views to allow the user the detection of interesting patterns. For example, when the user selects a data cluster from the Data Aggregation List (cf. Figures 9 and 8), respective data points are highlighted in the Time Series Path Map and the Color Time Bar. Thus, the user has three different scopes for the exploration of the selected data: (a) the distribution of the data on the Time Series Path Map, (b) occurrences of data elements along the time line in the Color Time Bar and (c) cluster value distributions in the Data Aggregation List (cf. Figure 8).

By means of transparency and plotting size, the user can counter over-plotting on his own by reducing the visibility of elements that are not selected.

Rollercoaster Animation

The Color Time Bar also contains a *Time Slider* for animated time series analysis. We can drag the Time Slider to a specific point or interval in time, and corresponding subsequences are highlighted with circular shapes in real-time on the Time Series Path Map. A schematical demonstration of our so called "Rollercoaster Animation" is given in Figure 4, an application is shown in Figure 10. This interactive animation allows a detailed exploration of the distribution of projected values over time, and also to detect periodic patterns on the Time Series Path Map. The latter is especially helpful in case of over-plotted displays, where a large amount of data elements is visualized on the display.

5 CASE STUDY

We apply our system to a data set from earth observation research. Based on consultation with domain researchers, we explore weather phenomena hidden in the

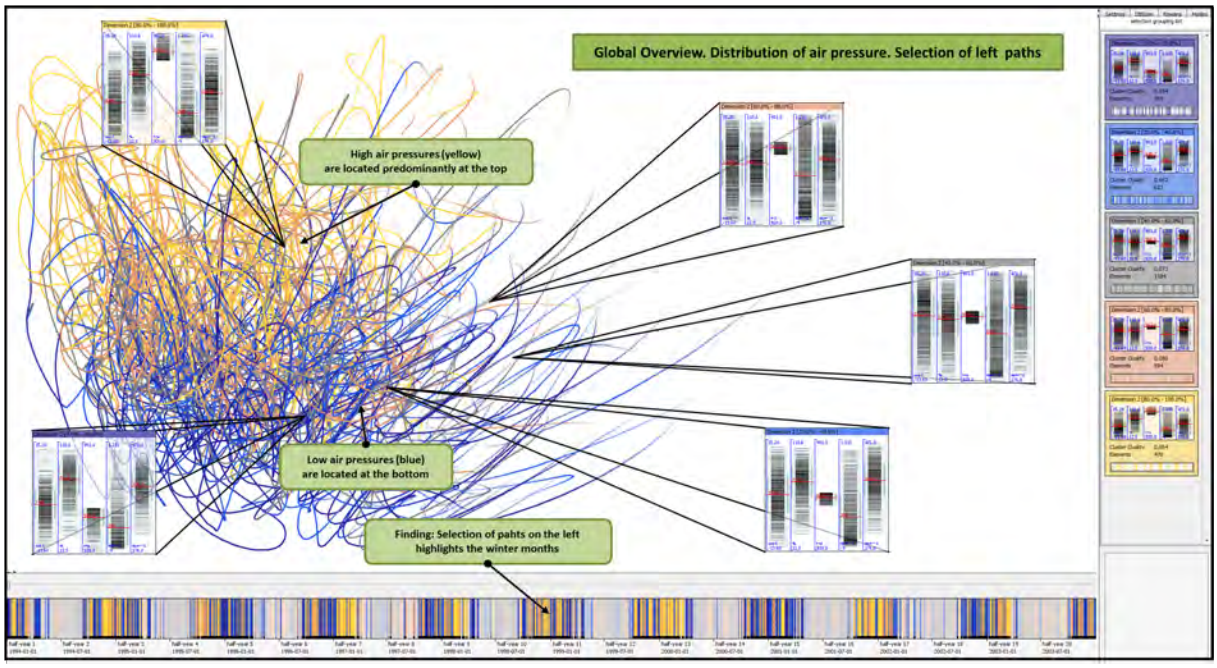


Figure 7: Data aggregation on single input data dimensions: Air pressure development. Selection of the left half of the paths (winter weathers). We discover a color gradient from high (top, yellow) to low (bottom, blue) air pressure values.

data like periodic patterns, frequent 'weather states' and abnormal behavior that can be found with our system.

5.1 Data Set and Application Domain

Our considered data set is acquired from the open data repository PANGAEA [21], operated by the Alfred Wegener Insitute (AWI) for Polar and Marine Research in Bremerhaven. PANGAEA archives and publishes geo-referenced scientific earth observation data in the research areas of water, ice, sediment and atmosphere. Our data set focuses on atmospheric weather measurements, gathered in the scope of the Baseline Surface Radiation Network (BSRN) [20] PANGAEA compartment. These measurements are multivariate atmospheric observations of radiation-related physical parameters) which were recorded every minute. We focus on a dataset of ten years duration, originated from the BSRN station in Payerne (Switzerland) in the time period of January 1st, 1994 to December 31th, 2003 [4]. Payerne is located in the center of the triangle Lyon, Milan and Fribourg at 491 meters above sea level. The climate of Payerne is temperate, semi-continental with average minimum temperatures at about -2°C in January and about 18°C in July. The average daily sunshine duration varies between 2 hours in January and 8 hours in July. Hence, the researchers affirm a yearly climate periodicity to the data that serves as ground truth and primary analysis goal. Beyond that, the so called "summer of the century" in 2003 produced temperature values up to 40°C and motivates us finding this and yet other anomalies in the data set.

We consulted researchers from BSRN to select a suitable parameter subset for detecting interesting weather scenarios. Besides *temperature*, *relative humidity* and *air pressure*, we incorporate the *short-wave downward radiation (SWD)* and the *long-wave downward radiation (LWD)*. The SWD is well suited to give statements about cloud occurrences. Most radiation is measured at the so called clear-sky condition, even when there are no clouds in the sky. It is used for climate research in general and in applied sciences, e.g., in land surface assimilation models, surface energy budget models, and ocean assimilation models. In agriculture, the short-wave downward radiation is used as an input for crop modeling and the solar industry applies it for estimations where to build solar power plants. The LWD is another important factor in the energetic exchange between atmosphere and the earth surface. While the solar dependent short-wave downward radiation is near zero at night, the long-wave downward radiation can be measured all night long. The long-wave downward radiation is higher when the sky is clear. By applying these five measurements as our data set, we are able to make statements about different weather states that possibly change within a seasonal cycle.

Due to the long time period of ten years, we determine each single day as one data point, periodic behaviors within single days are also discovered in the data set and possible to analyze with our system, but not in the focus in this case study. In order to remain on a uni-

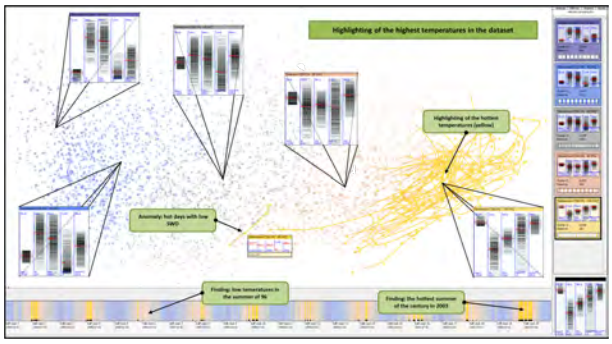


Figure 8: Advanced exploration of Figure 1. We have no problems in identifying the periodic appearance of hot temperatures in the summers in the Color Time Bar. Furthermore, the "summer of the century" anomaly in 2003 stands out with a lot of hot days.

versally accepted approach, we use a generic moving average routine to produce meaningful daily representatives, climate specific daily data aggregation procedures are not applied. Missing value periods of less than 6 hours are linearly interpolated, longer data gaps are ignored. We want to point out that other preprocessing approaches are possible and merely implicate for us the effort of reconfiguring parameters or, if necessary, add a domain-specific preprocessing routine.

5.2 Obtaining a Global Overview

We primarily obtain a global overview to the Time Series Path Map and the data, respectively (cf. Figures 1 and 6). This is crucial due to the described problems in dealing with large multivariate data and projection-based approaches (cf. Section 3). The Color Time Bar indicates a meaningful periodicity with in the seasonal cycle. We constitute Payernes climate to be warm in the summer and cold in the winter period (cf. Figure 1). The overview is completed with Figure 6, where the relative humidity appears to be high (rainy) on the left and low (dry) on the right. At least since the Time Color Bar shows summers to be dry and winters to be wet, we can constitute that the left half of the Time Series Path Map depicts the winter period whereas the summer time is allocated at the right of the display. We prove this hypothesis in Figure 7 by selecting the left half of the time series paths and obtain a meaningful segmentation on the Color Time Bar between summer and winter. Taking the cluster glyphs of the three discussed images into account, we assess correlations between dimension 1 (temperature), 4 (SWD) and 5 (LWD) and thus register another finding in the data set.

After we have received a global overview to the data and our views (some findings may appear evident to the reader so far), we now proceed our case study and focus on the exploration of more particular findings.

5.3 Findings in the Data Set

We now focus on abnormal behavior and anomalies in the data set. We try to discover the "summer of the century" of 2003 as a first finding. We use the view shown in Figure 1 and select the hottest data cluster (yellow); the result is shown in Figure 8. Besides, we discover the coldest summer of the data set in the year 1996 as a new finding. Together with the researchers from AWI, we find our final data exploration goal in the detection of thunderstorms and intense low-pressure systems. Besides the researchers expertise, we consult Internet forums, weather history literature, and insurance damage reports to verify our findings. Figure 9 displays our course of exploration. We focus on the air pressure dimension and apply our *gradient-based* statistical property that measures value changes over time. An aggregation to six clusters produces one group of about 200 highlighted data points that manifest extremely decreasing air pressure gradients. We tooltip a collection of five proven hurricanes and chose the most prominent and devastating hurricane *Lothar* for a detail on demand exploration. Figure 10 details about the air pressure development over 10 days in december 1999. The Rollercoaster Animation helps us navigating through a clearly arranged display, released from visual clutter and overview problems.

6 DISCUSSION

One of the most traditional visualization types for time series data are line charts. In case of multivariate time series, multiple parallel line charts can be used for data visualization. Eventually, projection-techniques such as studied in this paper need to be compared against alternative time series visualization options. While we have not yet done a formal comparison, we provide a conceptual discussion to point at possible advantages and disadvantages of the projection-based approaches vs. line chart approaches.

First, we expect the traditional line chart approach to have scalability problems with respect to very long time series, and to a high number of dimensions. The projection-based approach for the visualization of time series data aims at improving scalability with respect to (1) the time axis (long time series) and (2) the potentially high number of dimensions. Considering (1), information loss occurs for line charts as soon as the number of time stamps becomes larger than the number of available pixels on the x-axis of the line chart display. Basically, three observations can be made:

1. Drawing multiple data points per pixel coordinate leads to visual artifacts and information loss.

2. Downsampling the number of time stamps reduces the amount of information of the visualization.

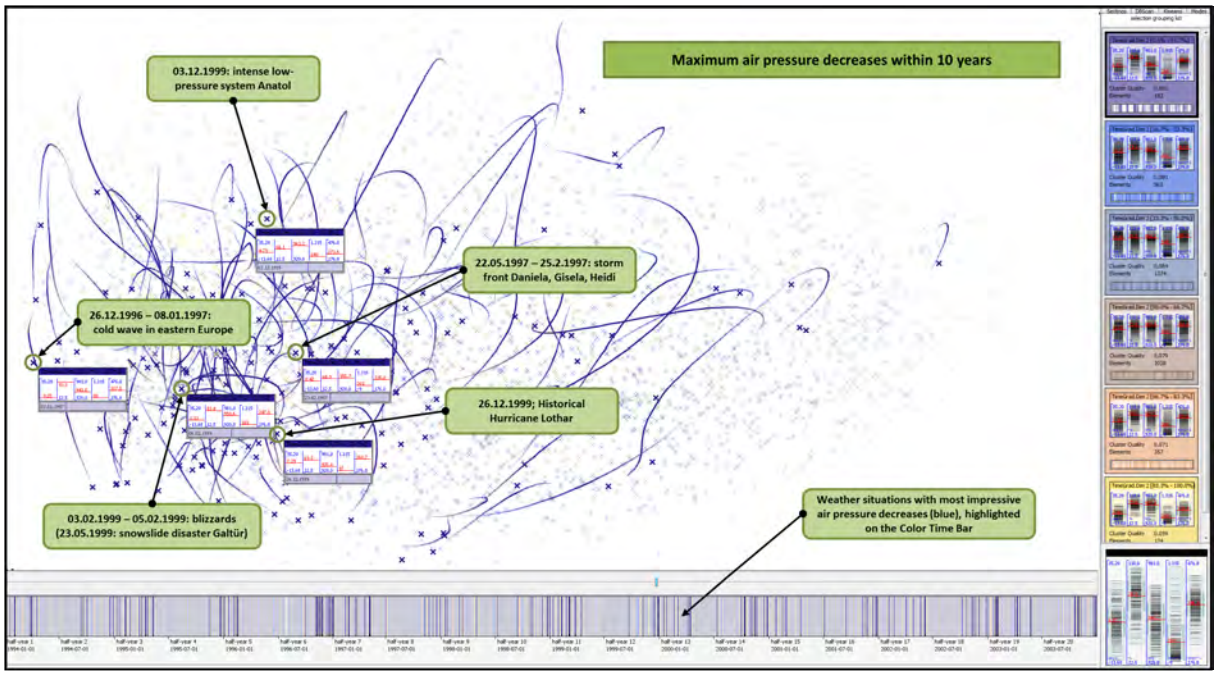


Figure 9: Detection of historic thunderstorms by highlighting most crucial air pressure decreases (blue).

3. Scrolling facilities allow to navigate large time series, yet can be cumbersome for very large time series and may lead to loss of user overview.

Using a 2D projection has the potential to show longer time series in a given display space, if an appropriate projection can be found. On the other hand, interpretation of the projected time series paths may become harder, as directions and distances cannot be read as straightforward as in a line chart.

Considering (2), a high number of dimensions may constitute a problem for multiple line charts. At some point, the available display space is exhausted when too many dimensions are combined in a multiple line chart visualization. In projection, dense data point regions are not only visual clutter. These regions represent dense regions in the input data space and offer potential starting points for further user search and filtering.

The second distinction between multiple line charts and projection concerns the number of data attributes to show. The projection condenses the information of all dimensions in one time series path, providing dimensionality reduction. In general, projection of multivariate data brings up questions about the application-dependent choice of the projection variant (cf. Subsection 3.1) and the preservation of information hidden in the input data. As future work, we need to compare the information preservation of multiple line charts (considering problems for large data or many dimensions) and projection-based time series visualization approaches. One first idea is to define a benchmark data set with periodic behavior that is compared in multiple line charts and in projection-based visualization. At present, we depict that the first two main components of the PCA-based 2D projection approach preserve 78% of the chosen 5D input data information in our weather data case study. Thus, we may assume that the amount of used information is rather high. Yet, more precise evaluation and comparison of the information contents and usage in parallel line charts and in projection-based approaches is needed.

7 CONCLUSION

We presented a system for the analysis of multivariate time-series data. The identification of relations between multiple attributes is an intrinsically difficult problem, even with a viable projection to 2D-space. In order to make such a visualization understandable for domain-experts, our system provides methods for statistical aggregation and clusterings, which can be steered by the user in a very flexible way. Beyond just showing cluster IDs we propose a new glyph-based visualization. This glyph shows the multivariate profiles of the clusters and allows for an effective comparison and interpretation of their properties. The system provides linked views to relate different perspectives of the data to each other. In cooperation with earth observation researchers, we tested the usefulness of the approach with a dataset for atmospheric weather measurements over a ten-years time frame.

We believe that the approach presented in this paper is easily applicable to time-series of different domains. In future projects we will apply and test this system with consumption data of the electric power grid. We used

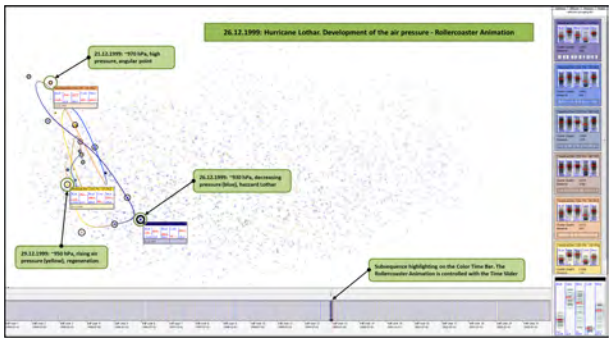


Figure 10: Rollercoaster Animation on hurricane Lothar. Air pressure coloring (blue means low).

projection techniques as an overview because of their popularity as a method for multivariate analysis. However, the methods to calculate, steer and explore the clusters are not restricted to a specific type of overview. In future, we will extend the linked views by other visualizations for multivariate time-series to test for the most effective combination of domain, overview and aggregation methods.

ACKNOWLEDGMENTS

We thank the Alfred Wegener Insitute (AWI) in Bremerhaven, particularly Rainer Sieger, Hannes Grobe and Gert König-Langlo, and everyone involved with PANGAEA for supporting this research effort.

8 REFERENCES

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer-Verlag New York Inc, 2011.

[2] Mihael Ankerst, Daniel A. Keim, and Hans-Peter Kriegel. Recursive pattern: A technique for visualizing very large amounts of data. In *Proceedings of Visualization '95, Atlanta, GA*, pages 279–286, 1995.

[3] Jürgen Bernard, Jan Brase, Dieter W. Fellner, Oliver Koepler, Jörn Kohlhammer, Tobias Ruppert, Tobias Schreck, and Irina Sens. A visual digital library approach for time-oriented scientific primary data. In *Proc. European Conference on Digital Libraries*, pages 352–363, 2010.

[4] Jürgen Bernard, Nils Wilhelm, Maximilian Scherer, Thorsten May, and Tobias Schreck. Reference list of 120 datasets from time series station payerne used for exploratory search. doi:10.1594/pangaea.783598, 2012.

[5] Lior Berry and Tamara Munzner. Binx: Dynamic exploration of time series datasets across aggregation levels. In *Proc. IEEE Symposium on Information Visualization*, 2004.

[6] D. Bhattarai and B.B. Karki. Visualization of atomistic simulation data for spatio-temporal information. In *The 14th Int'l. Conf. on Central Europe in Computer Graphics, Visualization and Computer Vision (WSCG'06)*, 2006.

[7] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000.

[8] Tim Dwyer and David R. Gallagher. Visualising changes in fund manager holdings in two and a half-dimensions. *Information Visualization*, 3:227–244, December 2004.

[9] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, Dominik Morent, and Jörn Schneidewind. Intelligent visual analytics queries. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 91–98, 2007.

[10] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. Importance driven visualization layouts for large time-series data. In *Proc. IEEE Symposium on Information Visualization*. IEEE Computer Society, 2005.

[11] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[12] Yueqi Hu, Shuangyuan Wu, Shihong Xia, Jinghua Fu, and Wei Chen 0001. Motion track: Visualizing variations of human motion data. In *PacificVis*, pages 153–160, 2010.

[13] N.K. Kasabov and Q. Song. Denfis: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *Fuzzy Systems, IEEE Transactions on*, 2002.

[14] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Data Mining, Fifth IEEE International Conference on*, pages 226 – 233. Ieee, 2005.

[15] Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, 3rd edition, 2001.

[16] Zoltan Konyha, Kresimir Matkovic, Denis Gracanin, Mario Jelovic, and Helwig Hauser. Interactive visual analysis of families of function graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1373–1385, November 2006.

[17] J. Lin, E. Keogh, S. Lonardi, J.P. Lankford, and D.M. Nystrom. VizTree: a tool for visually mining and monitoring massive time series databases. In *Proc. of the int. conf. on Very Large Data Bases*, pages 1269–1272. VLDB Endowment, 2004.

[18] Yi Mao, Joshua Dillon, and Guy Lebanon. Sequential document visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13:1208–1215, 2007.

[19] Fabian Mörchen and Alfred Ultsch. Efficient mining of understandable patterns from multivariate interval time series. *Data Min. Knowl. Discov.*, 15(2):181–215, 2007.

[20] A. Ohmura, E. G. Dutton, B. Forgan, C. Fröhlich, H. Gilgen, H. Hegner, A. Heimo, G. König-Langlo, B. mcarthur, G. Müller, R. Philipona, R. Pinker, C. H. Whitlock, K. Dehne, and M. Wild. Baseline surface radiation network (BSRN/WCRP): New precision radiometry for climate research. *Bull. Amer. Met. Soc.*, 79:2115–2136, 1998.

[21] PANGAEA - Data Publisher for Earth and Environmental Science. http://www.pangaea.de/. Last accessed on April 5, 2012.

[22] Y. Sakamoto, S. Kuriyama, and T. Kaneko. Motion map: image-based retrieval and segmentation of motion data. In *Proc. 2004 ACM SIGGRAPH/Eurographics symposium on computer animation*. Eurographics Association, 2004.

[23] Geoffroy Simon, Amaury Lendasse, Marie Cottrell, and Université Paris. Long-term time series forecasting using self-organizing maps: the double vector quantization method, 2003.

[24] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, Cheshire, CT, USA, 1986.

[25] Jarke J. Van Wijk and Edward R. Van Selow. Cluster and calendar based visualization of time series data. In *Proc. IEEE Symposium on Information Visualization*, pages 4–9. IEEE Computer Society, 1999.

[26] Matthew O. Ward and Zhenyu Guo. Visual exploration of time-series data with shape space projections. *Eurographics / IEEE Symposium on Visualization (EuroVis)*, 30(3), 2011.

[27] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *proceedings of the IEEE Symposium on Information Visualization*, pages 7 – 13, 2001.

[28] H. Ziegler, M. Jenny, T. Gruse, and D.A. Keim. Visual market sector analysis for financial time series data. In *Visual Analytics Science and Technology (VAST), IEEE Symposium on*, pages 83–90, 2010.

Design and Analysis of Visualization Techniques for Mobile Robotics Development

Alex Kozlov
The University of Auckland, New Zealand
akoz002@aucklanduni.ac.nz

Bruce A. MacDonald
The University of Auckland, New Zealand
b.macdonald@auckland.ac.nz

Burkhard C. Wünsche
The University of Auckland, New Zealand
burkhard@cs.auckland.ac.nz

ABSTRACT

Simultaneous localization and mapping (SLAM) algorithms are of vital importance in mobile robotics. This paper presents novel Augmented Reality (AR) visualization techniques for SLAM algorithms, with the purpose of assisting algorithm development. We identify important algorithm invariants and parameters and combine research in uncertainty visualization and AR, to develop novel AR visualizations, which offer an effective perceptual and cognitive overlap for the observation of SLAM systems. A usability evaluation compares the new techniques with the state-of-the-art inferred from the SLAM literature. Results indicate that the novel correlation and color-mapping visualization techniques are preferred by users and more effective for algorithm observation. Furthermore the AR view is preferred over the non-AR view, while being at least similarly effective. Since the visualizations are based on general algorithm properties, the results can be transferred to other applications using the same class of algorithms, such as Particle Filters.

Keywords

Algorithm visualisation, augmented reality, robot programming, human-robot interfaces

1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) [SSC90, LDW91] is a popular and important class of estimation algorithms, addressing the challenge of autonomous map-building for mobile robots. A robot must have a model, or a map, of the physical environment in order to carry out useful navigation tasks. The robot must additionally localize itself within the environment. In SLAM the robot autonomously explores and maps its environment with its sensors while localizing itself at the same time. Despite considerable research, open challenges in SLAM include implementations in unstructured, difficult, and large scale environments [BFMG07], multi-robot SLAM [NTC03] as well as SLAM consistency and convergence [MCDFC07].

SLAM development is made more difficult by its probabilistic nature. In SLAM, neither the robot location nor the environment map are known in advance. However, in order to map the environment the robot location needs to be known with accuracy, and in order to localize the robot the environment map needs to be known with accuracy. Visualizations aid the development and testing of SLAM algorithms by revealing relationships between robot and algorithm states and environmental parameters. Existing SLAM visualization techniques are purely virtual and limited to basic state information, thus lacking *perceptual* and *cognitive* overlap between the robot and the human developer [BEFS01].

Augmented Reality (AR) involves spatially registering virtual objects in real time within a view of a real scene [BR05, Azu97]. AR has been used in robotics to enhance the human-robot interface, but has never been applied to SLAM visualization. This paper presents and evaluates novel AR visualization techniques for SLAM with the purpose of assisting algorithm development and debugging. The introduced concepts and lessons learned are applicable to other estimation algorithms in robotics and related fields.

Section 2 outlines the SLAM algorithms for which the visualizations have been developed. Section 3 presents a review of fields we draw on. Section 4 summarizes the visualization requirements. Section 5 explains the visualization techniques. Section 6 presents the evaluation of the visualizations, and section 7 concludes our paper.

2 SLAM BACKGROUND

The two most popular SLAM algorithm archetypes, the Extended Kalman Filter (EKF) SLAM [GL06, DWB06] and FastSLAM [MTKW03, Mon03], are both based on recursive Bayesian estimation. These

algorithms were targeted for visualization because they are the two most important and prevalent SLAM solution methods [DWB06]. The visualizations presented in this paper would also be relevant for any modified Kalman Filter or Particle Filter based algorithm.

2.1 EKF SLAM

In feature-based EKF SLAM the state is represented by a multivariate Gaussian distribution with mean x and covariance P:

$$x = \begin{bmatrix} x_r \\ x_{f_1} \\ \vdots \\ x_{f_n} \end{bmatrix} \tag{1}$$

$$P = \begin{bmatrix} P_r & P_{r,f_1} & \dots & P_{r,f_n} \\ P_{f_1,r} & P_{f_1} & \dots & P_{f_1,f_n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{f_n,r} & P_{f_n,f_1} & \dots & P_{f_n} \end{bmatrix} \tag{2}$$

x_r is the estimated robot pose and $x_{f_i}, i = 1,\dots,n$ is the estimated position of an environment feature f_i. The main diagonal elements $P_r, P_{f_1}, \dots, P_{f_n}$ are error covariance matrices of the robot pose and the landmark locations. The off-diagonal elements are cross-covariance matrices between the robot and feature positions. The recursive estimation steps of the algorithm are *motion prediction*, *data association*, *observation update* and *feature initialization*.

2.2 FastSLAM

In FastSLAM, which is based on Rao-Blackwellized Particle Filters, the state is represented by a set of N particles:

$$\{w^{(i)}, X_r^{(i)}, \mu_{f_1}^{(i)}, \dots, \mu_{f_n}^{(i)}, \Sigma_{f_1}^{(i)}, \dots, \Sigma_{f_n}^{(i)}\}_i^N \tag{3}$$

where for particle i, $w^{(i)}$ is the particle weight, $X_r^{(i)}$ is the sampled robot path, and each map feature f_j is represented independently by a Gaussian distribution with mean $\mu_{f_j}^{(i)}$ and covariance $\Sigma_{f_j}^{(i)}$. The recursive estimation steps of the algorithm are *motion prediction*, *weight update*, *resampling* and *map update*.

3 LITERATURE REVIEW

A number of *Robotics Development Environments (RDEs)* are available for use in robotics programming, but none offers visualizations for SLAM. Examples include Player [GVS+01], CARMEN (Carnegie Mellon robot navigation toolkit) [MRT] and Pyro (Python Robotics) [BKMY03]. These offer purely virtual sensor data visualizations. Possibly the most flexible support for visualizations is offered by ROS (Robot Operating System) [QGC+09], which includes a variety of data-types such as point clouds, geometric primitives, robot poses and trajectories.

No formal studies have been done for visualization techniques in SLAM. The SLAM visualization state of the art is inferred from the visual representations of SLAM systems and data in the literature. The current "conventional" method of visualizing EKF-style SLAM is by showing the mean estimates for the robot and features, along with the covariance ellipsoids showing the individual uncertainties (e.g. [BNG+07, NCMCT07]). For Particle Filter type SLAM, all current robot poses and mean feature locations are shown for all particles (e.g. [MTKW03, Mon03]). Perhaps the most interesting example of an existing SLAM visualization is the 3D graphical representation of the robot in relation to the mapped obstacles, with the uncertainties shown by dotted lines around the objects [NLTN02]. Martinez-Cantin et al. visualized a constructed 3D map from the robot's point of view [MW03].

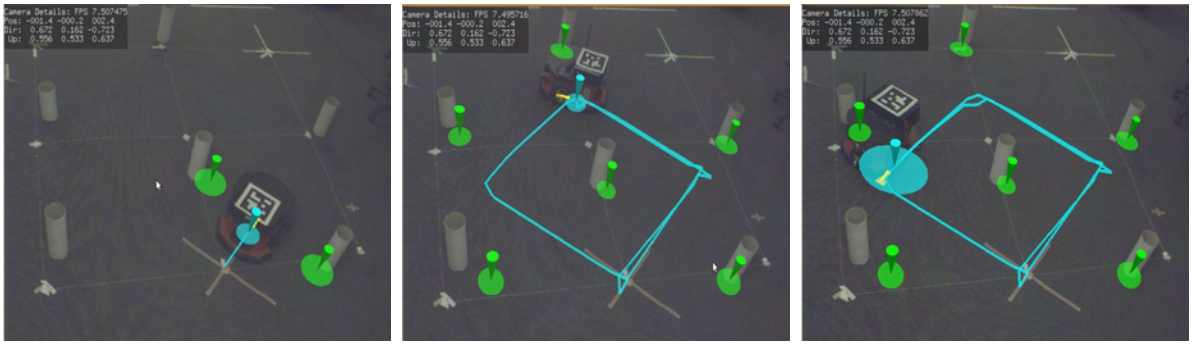
None of the basic SLAM visualizations suggested so far employs an AR environment. However, AR systems have been developed and used in robotics. An example is *ARDev* [CM06, CM09]. It provides perceptual overlap between the human and the robot by visualizing sensor data within the robot's real world environment. Nunez et al. use AR for supervision of semi-autonomous robot navigation tasks [NBPLS06]. A topological map is generated online and visualized with AR. Daily et al. use AR to supervise a swarm of 20 robots for search and rescue scenarios [DCMP03]. Virtual arrows above every swarm member in view convey the intention and direction of travel. AR has also seen considerable application in mobile robot tele-operation [BOGH+03] and manipulator robotics [NCE+07].

4 VISUALIZATION REQUIREMENTS

The underlying requirement for all of the visualizations is to embed information within the context of the real world environment the mobile robot operates in. This provides a qualitative comparison between the estimates and the ground truth, and shows sources of potential errors within the real environment.

4.1 EKF SLAM Requirements

The fundamental EKF SLAM requirement is to visualize the state and the individual uncertainty covariances. The state consists of 2D robot and feature locations, and the robot orientation in the ground plane. The 2 by 2 covariance matrices for the robot and each feature indicate the positional uncertainty, together with the robot orientation variance.



(a) (b) (c)

Figure 1: **Conventional AR EKF Visualization:** EKF-SLAM state and covariance visualization in AR, showing progression over time. The AR view provides a direct comparison of the estimates against the ground truth in the real world environment; this shows the discrepancies between the truth and the estimates.

Correlations between features are well known to be important for correct SLAM operation [DWRN96]. In [DNC+01] feature correlations are defined as follows. Let d_{ij} be the relative position between any two feature estimates f_i and f_j. Let $P_{d_{ij}}$ be the covariance of d_{ij} as follows:

$$d_{ij} = x_{f_i} - x_{f_j} \tag{4}$$

$$P_{d_{ij}} = P_{f_i} + P_{f_j} - P_{f_i,f_j} - P_{f_i,f_j}^T \tag{5}$$

$P_{d_{ij}}$ is a measure of relative error between the two features, and is therefore also a measure of their correlation. The expected convergence property [HD07, DNC+01] is that correlations strengthen as observations are made over time, i.e. the volume of uncertainty in $P_{d_{ij}}$ is non-increasing. Visualization of the correlation behaviour is essential. Violations of this behaviour (i.e. weakening correlations) indicate problems in the SLAM system, and therefore must be detected by the user. Specifically, the 2 by 2 covariance matrix $P_{d_{ij}}$ must be visualized for all feature pairs, together with its non-increasing volume of uncertainty trend. Violations must be exemplified.

4.2 FastSLAM Requirements

The fundamental FastSLAM requirement is to visualize the state represented by the set of particles. This means visualizing 2D points for the robot and Gaussian mean feature locations, for all particles. Additionally robot orientations for all particles must be visualized.

Due to the Rao-Blackwellized structure of FastSLAM, sampling is only done on the robot path. The error in the map estimation for a given particle is dependent on the accuracy of the robot path. For this reason, it is important to visualize the relationship between the robot path and map estimates within particles, or intra-particle associations. Specifically, this refers to visually linking estimates from the same particle, and distinguishing these from other particles. Visualization of the individual weight for each particle is also important in order to gain insight into the resampling phase of the algorithm.

Lastly, a more qualitative and more intuitive representation of the SLAM solution produced by the filter would be useful. This needs to show a better overall picture of the solution, possibly at the cost of lower information content or being less exact.

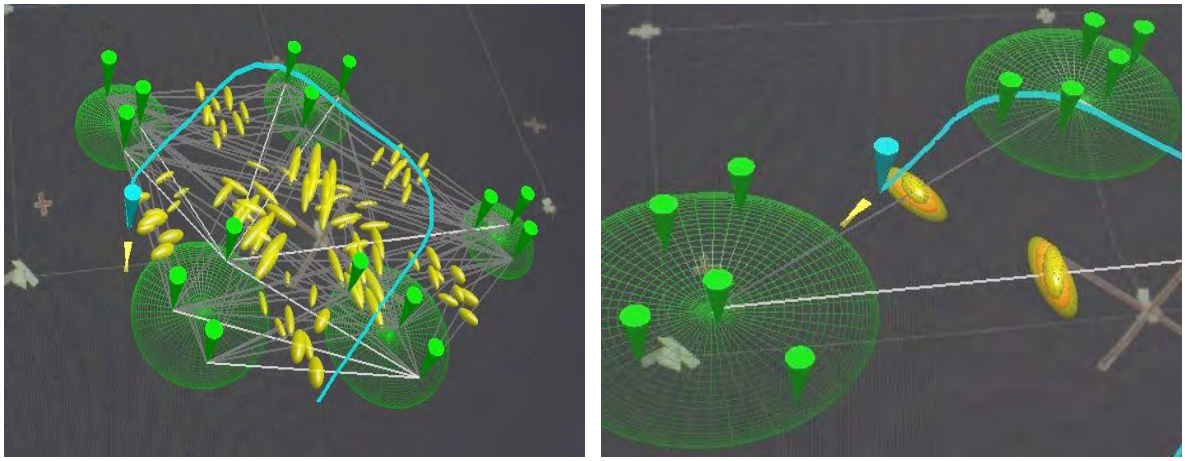
5 AR VISUALIZATION TECHNIQUES

5.1 EKF SLAM Visualizations

5.1.1 Conventional EKF SLAM Visualization

Fig. 1 presents the state-of-the-art conventional EKF visualization implemented in AR. The underlying real world images present an overhead view of the robot and its environment. The robot is a PIONEER 3-DX [Rob08]. The map the robot is building consists of two dimensional points in the ground plane represented by white cardboard cylinders. The cylinders are the only physical objects being mapped and are extracted from raw laser rangefinder data. The robot drives around and performs SLAM in a small 1 by 1 meter loop. The graphical icons augmenting the video frames represent SLAM information:

- **Cyan Marker** - The cyan downward pointed cone represents the estimated robot position. The cyan ellipsoid underneath is the robot position covariance. The cyan line is the robot path.
- **Green Marker** - The green downward pointed cone represents the estimated feature position. The green ellipsoid underneath is the feature position covariance.
- **Yellow Marker** - The yellow triangular pointer represents the robot orientation estimate. A semitransparent yellow circular wedge represents the orientation variance.



(a) (b)

Figure 2: **EKF SLAM Correlations Clustering.** (a) shows all inter-cluster correlations, (b) shows only the maximum, mean, and minimum inter-cluster correlations. Green wireframe circles represent spatial clusters.

The markers represent the SLAM estimates for qualitative real-time visual comparison against the ground truth presented in the real image, i.e. the green markers correspond to the white cardboard cylinders and the blue marker to the physical robot. For the orientation an "arrow" type marker was chosen, as commonly used in SLAM and robotics visualizations. For the covariance, the common tensor ellipsoid glyph was used, which is superior to line or arrow type ellipsoid glyphs. Colour was used to define specifically what the estimate refers to, i.e. robot or features. The design follows the "Natural Scene Paradigm", which is based on humans' ability to immediately perceive complex information in a natural scene [WL01].

5.1.2 Correlations Visualization

In previous work [KMW09] we presented the novel feature correlation visualization shown in Fig. 2a. It satisfies the requirements discussed earlier. For every pair of features $\{f_i, f_j\}$ the visualization contains:

- A line linking the feature estimates f_i and f_j
- A yellow tensor "correlation ellipsoid" for $P_{d_{ij}}$ rendered at the half-way point on the line

As $P_{d_{ij}}$ is a two-dimensional covariance, it produces a 2D tensor ellipsoid. However, the problem of visual cluttering becomes evident when many such ellipsoids grow in size and overlap. It becomes difficult to discern any individual ellipsoids. To mitigate this issue the ellipsoids were inflated to a shaded 3D shape. The second eigenvalue is used for the length of the axis into the third dimension. Giving a 3D shaded volume to the correlation ellipsoids provides better visual distinction to overlapping ellipsoids, however a limitation of this method is that it occludes more of the background world image.

Strengthening correlations show up as decreasing volumes of the correlation ellipsoids. If the volume of the correlation ellipsoid *increases* (i.e. the correlation weakens), this is considered a violation of the expected behaviour. This occurrence is exemplified in the visualization by changing the colour of the ellipsoid to red. Thus, the visualization allows the observation of the expected correlations trend, and the detection of its violations.

The problem of visual cluttering is resolved by *spatial clustering* using the standard single-linkage hierarchical clustering method [LL98]. Features are divided into spatial clusters, and only the correlations between features in different clusters are shown. The green wireframe circles exemplify the clusters computed by the algorithm. This image demonstrates a pure virtual simulation of the SLAM algorithm, and hence no physical robot and environment is shown.

In order to further reduce the number of correlations in view the user can select to only see the minimum (yellow), mean (orange), and maximum (yellow) inter-cluster correlations (Fig 2b). The expected correlation convergence can be observed through the non-increasing size of the minimum correlation ellipsoid [Koz11].

5.2 FastSLAM Visualizations

5.2.1 Conventional FastSLAM Visualization

Fig. 3 presents the conventional state-of-the-art FastSLAM visualization implemented for the first time in AR. The underlying real world images present an overhead view of the robot and the environment the robot is working in. The graphical icons augmenting the video frames represent SLAM information:

(a) (b) (c)

Figure 3: **Conventional FastSLAM AR Visualization:** particle representation for the robot pose and features, showing the joint SLAM posterior distribution computed by the particle filter and its progression over time.

- **Cyan Marker** - The cyan downward pointed cone represents the sampled robot location for a given particle.
- **Yellow Marker** - The yellow arrow-type marker represents the sampled robot orientation for a given particle.
- **Green Marker** - The green downward pointed cone represents a Gaussian mean feature location for a given particle.

The visualization shows the joint SLAM posterior probability distribution of the robot pose and the feature map. As for the EKF, the markers represent the SLAM state for qualitative visual comparison against the ground truth presented in the real image, i.e. the green markers correspond to the white cardboard cylinders and the blue marker to the physical robot.

5.2.2 *Colour Mapping Visualizations*

Fig. 4 presents a colour-mapping visualization technique addressing the requirements of intra-particle associations and particle weights, as discussed earlier. First the centroid and maximum distance from the centroid are computed for the current robot positions in the particles. This creates a local polar coordinate frame for the robot positions (and thus the particles they belong to), originating at the centroid. Then each particle's polar coordinates are mapped directly onto the Hue and Saturation parameters in the HSV colour model. Thus, each particle which has a unique robot position is assigned a unique colour. This colour is used to encode members of the same particle (intra-particle associations), e.g. a red feature and a red robot pose belong to the same particle. This shows the important relationship between the map and robot path estimations. In the final step, the particle weight is encoded into the Value (brightness) parameter of the HSV model. Lighter coloured markers indicate higher weights, and darker colours indicate lower weights. Fig. 5 shows the

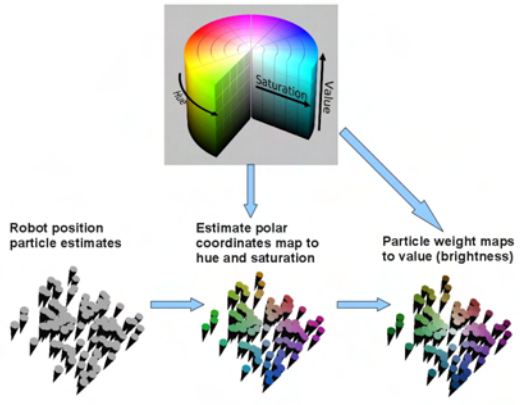


Figure 4: **The colour-mapping technique.**

colour-mapping visualization applied in SLAM. The colour-coded relationship between the robot position and feature errors is clearly visible.

5.2.3 *Surface Density Visualization*

Fig. 6 presents a novel surface density visualization technique developed for FastSLAM. The purpose of this visualization is to present a better overall qualitative picture of the SLAM solution produced by the filter. Here the joint SLAM posterior probability of the robot and the features is represented by a smooth, shaded 3D surface. The mapping area is divided into an uniform (customizable) grid, where the density of a given cell is given by the number of particle members (robot pose or features) within that cell. Then the surface is interpolated over the grid using a *Radial Basis Function (RBF)* [Buh03], with the density giving the surface height. If colour-mapping is enabled, the colour for each cell is the average of the particle colours within it. Otherwise, the cyan surface represents the robot pose and the green surfaces the features. In addition, a single arrow is drawn above each cell of the robot pose surface. This is the average robot orientation within the cell.

(a) (b) (c)

Figure 5: **Intra-particle Associations Visualization:** colour-mapping used to show members belonging to the same given particle, showing progression over time. Brightness indicates particle weights.

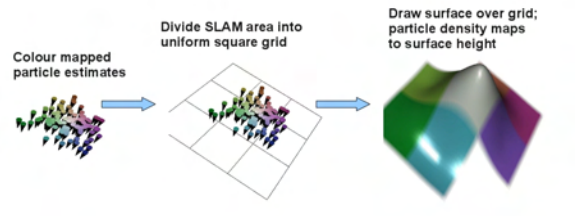


Figure 6: **The surface colour-mapping technique.**

Fig. 7 shows the surface density visualization without the colour mapping. Intuitively the height of the surface indicates the SLAM posterior probability. The shape of the surface provides a good qualitative picture of the uncertainty spread of the distribution, as compared to rendering each individual marker. Fig. 8 shows the colour-mapped surface density visualization. This offers the benefits of both the surface and the colour-mapping techniques. The visualization shows both the shape of the uncertainty spread and the colour-mapped intra-particle associations.

6 EVALUATION

6.1 Experimental Setup

The visualization system was implemented with a ceiling mounted Prosilica EC 1350C Firewire camera for image capture. Registration was done using ARToolKitPlus and a fiducial marker mounted on the robot. The robot's initial position is registered in the AR coordinate frame as the origin of the SLAM map. This allows the registration of the SLAM data in the AR coordinate frame. Videos were taken of the robots SLAM performance using different visualization techniques for correctly implemented SLAM algorithms and versions with typical errors we inferred from the literature and a previous user survey [Koz11].

6.2 Methodology

We performed five experiments summarised in table 1 in order to investigate the effectiveness of the visualizations for assisting SLAM development. In particular we evaluated AR-based visualisation techniques versus non-AR visualisation techniques and novel AR visualisation versus AR-implementations of techniques considered current state-of-the-art. The purpose of the study was to compare the effectiveness of the visualization techniques for SLAM algorithm development, i.e. fault detection and fault correction.

Fault Detection Experiments		
Experiment	**Vis 1**	**Vis 2**
EKF Exp 1	Conventional AR	Conventional non-AR
EKF Exp 2	Correlations AR	Conventional AR
FastSLAM Exp 1	Conventional AR	Conventional non-AR
FastSLAM Exp 2	Colour-mapping AR	Conventional AR
FastSLAM Exp 3	Surface density AR	Conventional AR

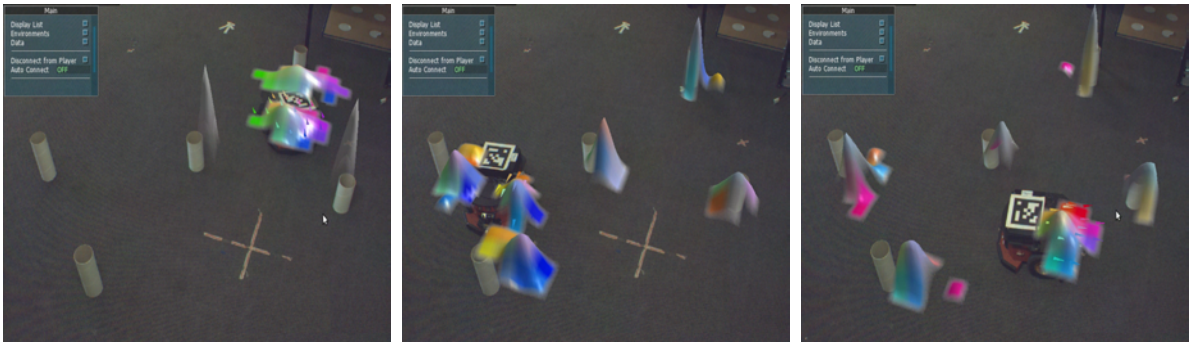
Table 1: Fault detection experiments summary. Each experiment compared Vis 1 with Vis 2.

The experiments were performed as a web-based survey questionnaire. Participants were invited over email, through the major international robotics mailing lists. These included Robotics Worldwide, Australian Robotics and Automation Association (ARAA) and European Robotics Research Network (EURON). Ideally the desired population of the participants would be SLAM developers; but in practice to obtain sufficient participants the population scope was widened to *robotics* developers. The experiments involved participants watching online videos of the visualizations and answering questions about the visualizations.

Within the questionnaire document, the concepts of SLAM and AR were first explained, along with in-

(a) (b) (c)

Figure 7: **Surface Density Visualization:** the surface density visualization without the colour-mapping, showing progression over time. The shape and height of the surface conveys the joint SLAM posterior distribution computed by the particle filter.



(a) (b) (c)

Figure 8: **Colour Mapped Surface Visualization:** the surface density visualization with the colour-mapping, showing progression over time. The visualization shows both the shape of the joint SLAM posterior distribution and the associations within particles.

troductory videos and explanations about the visualizations. Each AR visualization was presented with a video of it being used for SLAM with a real robot and cylindrical point features, along with a written explanation. To present the non-AR visualization, two videos were used. One was the virtual SLAM visualization, and the other was the video of the physical robot performing SLAM corresponding to the SLAM visualization.

After showing correct operation, artificial faults were introduced into the SLAM systems. Within each experiment the same fault was used to compare the visualizations, however the visualizations showed the fault in different ways. For each visualization, the participants were asked as a multi choice question what SLAM fault is present in the visualization (if any). For each pair of visualizations compared, the participants were also asked in a short answer question which visualization they felt was more effective (Vis 1, Vis 2, or neither) and why. Details of the study are found in [Koz11].

6.3 Results

There were 24 participants in the EKF evaluation, and 14 participants in the FastSLAM evaluation.

In EKF Exp 1 users detected 75% of errors with the AR visualization and 70% of errors with the non-AR visualization. Users liked that the AR visualization combined a view of the real environment with the SLAM information. Reasons for prefering non-AR were perception difficulties in AR due to the 3D camera perspective, deformation, depth and projection, and the real-world camera image. In FastSLAM Exp 1 all of the participants preferred the AR visualization. In terms of effectivness both visualisations resulted in 57% of errors being detected.

In EKF Exp 2 our new correlation visualization allowed users to detect 79% of errors, whereas the traditional visualization only allowed detection of 50% of errors. Users liked in the correlation visualization the explicit representation of correlation faults enabling a faster detection. Reasons for prefering the conventional AR visualization were clearer, more intuitive representation of robot pose/landmarks and faults therein, the correlation ellipsoids being hard to understand and occluding the landmark/robot markers, and robot/landmark covariances being more representative of the estimation process.

In FastSLAM Exp 2 users were able to detect 64% of errors using colour mapped particles and 35% of errors using the conventional visualization. Users liked about colour-mapping the clear representation of particle weighting and the resampling process, and that colour mapping offers more information in a compact representation allowing for better fault detection.

In FastSLAM Exp 3 users identified 42% of errors using the surface density visualization and 71% of errors using the conventional visualization. Users liked the compact and effective representation of the particle set distribution in the surface density AR visualization, and that the peak of the surface indicates the most likely estimate position whereas the spread shows the amount of divergence in the estimates. However, users complained that the surface representation is too opaque and obscures the true landmarks, and that the surface view does not show the robot orientation clearly. Users stated that the conventional AR visualization is easier to analyze in order to detect errors.

7 CONCLUSIONS

This paper presented novel AR visualization techniques for SLAM algorithms. The visualization requirements were challenging because SLAM algorithms are detailed, complex, and must address real world uncertainties. To address the requirements, visualizations were developed in AR to target the most important aspects of the SLAM algorithms, including feature correlations, particle weights and relationships.

Our Evaluation shows that AR visualizations are preferred over non-AR visualizations, and that the novel techniques for feature correlations is more effective than the existing state of the art for SLAM fault detection. The visualizations are effective because they target specific aspects of the algorithm and because AR enables visualization of the real world and associated uncertainties. The correlation visualization can be adapted to any application requiring representation of correlations between physical entities. Care must be taken that visualization icons do not obscure relevant real-world information in the camera view and that visual complexity does not put undue stress on the user. Hence small point based icons are preferable over more complex and information rich surface representations. The presented visualizations perform differently well for different types of errors. Ideally the user should be able to swap interactively between all of the presented techniques.

8 REFERENCES

[Azu97] R.T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BEFS01] C. Breazeal, A. Edsinger, P. Fitzpatrick, and B. Scassellati. Active vision for sociable robots. *IEEE Trans. Syst., Man, Cybern.*, 31(5):443–453, 2001.

[BFMG07] J.L. Blanco, J.A. Fernandez-Madrigal, and J. Gonzalez. A New Approach for Large-Scale Localization and Mapping: Hybrid Metric-Topological SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2061–2067, 2007.

[BKMY03] D. Blank, D. Kumar, L. Meeden, and H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing (JERIC)*, 3(4):1–15, 2003.

[BNG+07] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot. Consistency of the EKF-SLAM algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3562–3568. IEEE, 2007.

[BOGH+03] V. Brujic-Okretic, J.Y. Guillemaut, L.J. Hitchin, M. Michielen, and G.A. Parker. Remote vehicle manoeuvring using augmented reality. In *International Conference on Visual Information Engineering*, pages 186–189, 2003.

[BR05] O. Bimber and R. Raskar. *Spatial Augmented Reality : Merging Real and Virtual Worlds*. A K Peters, Limited, 2005.

[Buh03] M. Buhmann. *Radial basis functions theory and implementations*. Cambridge University Press, 2003.

[CM06] T.H.J. Collett and B.A. MacDonald. Developer oriented visualisation of a robot program. In *ACM SIGCHI/SIGART Human-Robot Interaction*, pages 49–56, 2006.

[CM09] T. H. J. Collett and B. A. MacDonald. An augmented reality debugging system for mobile robot software engineers. *Journal of Software Engineering for Robotics*, 1(1):1–15, 2009.

[DCMP03] M. Daily, Y. Cho, K. Martin, and D. Payton. World embedded interfaces for human-robot interaction. In *Annual Hawaii International Conference on System Sciences*, pages 6–12, 2003.

[DNC+01] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization

and map building (slam) problem. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 17(3):229–241, 2001.

[DWB06] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping: Part 1. *IEEE Robotics and Automation Magazine*, 13(2):99–108, 2006.

[DWRN96] H. Durrant-Whyte, D. Rye, and E. Nebot. Localisation of automatic guided vehicles. In *Robotics Research: The 7th International Symposium (ISRR'95)*, pages 613—-625, 1996.

[GL06] S. Ge and F. Lewis. *Autonomous mobile robots : sensing, control, decision- making, and applications*. Boca Raton, FL CRC/Taylor and Francis, 2006.

[GVS+01] B.P. Gerkey, R.T. Vaughan, K. Stoy, A. Howard, G.S. Sukhatme, and M.J. Mataric. Most valuable player: a robot device server for distributed control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1226–1231, 2001.

[HD07] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23(5):1036–1049, 2007.

[KMW09] Alex Kozlov, Bruce MacDonald, and Burkhard C. Wünsche. Covariance Visualisations for Simultaneous Localisation and Mapping. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2009)*, pages 1 – 10, December 2009. `http://www.cs.auckland.ac.nz/~burkhard/Publications/ACRA2009_KozlovMacDonaldWuensche.pdf`.

[Koz11] Alex Kozlov. *Augmented Reality Technologies for the Visualisation of SLAM Systems*. PhD thesis, The University of Auckland, 2011.

[LDW91] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localisation for an autonomous mobile robot. In *Proc. IEEE Int. Workshop Intell. Robots Syst. (IROS)*, pages 1442–1447, 1991.

[LL98] Pierre Legendre and Louis Legendre. *Numerical ecology*. New York : Elsevier, 1998.

[MCDFC07] R. Martinez-Cantin, N. De Freitas, and J.A. Castellanos. Analysis of particle methods for simultaneous robot localization and mapping and a new algorithm: Marginal-slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2415–2420, 2007.

[Mon03] M. Montemerlo. *FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association*. PhD thesis, Carnegie Mellon University, 2003.

[MRT] M. Montemerlo, N. Roy, and S. Thrun. *Carmen, Robot Navigation Toolkit*. Retrieved June 2007 from `http://carmen.sourceforge.net/`.

[MTKW03] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fast-slam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1151–1156, 2003.

[MW03] I. Mahon and S. Williams. Three-dimensional robotic mapping. In *Australasian Conference on Robotics and Automation (ACRA)*, 2003.

[NBPLS06] R. Nunez, J.R. Bandera, J.M. Perez-Lorenzo, and F. Sandoval. A human-robot interaction system for navigation supervision based on augmented reality. In *IEEE Mediterranean Electrotechnical Conference*, pages 441–444, 2006.

[NCE+07] A. Nawab, K. Chintamani, D. Ellis, G. Auner, and A. Pandya. Joystick mapped Augmented Reality Cues for End-Effector controlled Tele-operated Robots. In *IEEE Virtual Reality Conference*, pages 263–266, 2007.

[NCMCT07] J. Neira, J.A. Castellanos, R. Martinez-Cantin, and J.D. Tardos. Robocentric map joining: Improving the consistency of EKF-SLAM. *Robotics and Autonomous Systems*, 55(1):21–29, 2007.

[NLTN02] P. Newman, J. Leonard, J.D. Tardos, and J. Neira. Explore and return: experimental validation of real-time concurrent mapping and localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1802–1809, 2002.

[NTC03] J. Neira, J.D. Tardos, and J.A. Castellanos. Linear time vehicle relocation in SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 427–433, 2003.

[QGC+09] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[Rob08] ActivMedia Robotics. *PIONEER P3-DX*. Retrieved January 2008 from `http://www.activrobots.com/ROBOTS/p2dx.html`, 2008.

[SSC90] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[WL01] Burkhard C. Wünsche and Richard Lobb. A scientific visualization schema incorporating perceptual concepts. In *Proceedings of IVCNZ '01*, pages 31–36, 2001. `http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ01_WuenscheLobb.pdf`.

An Applied Approach for Real-Time Level-of-Detail Woven Fabric Rendering

Wallace Yuen
The University of Auckland,
New Zealand
wyue013@aucklanduni.ac.nz

Burkhard C. Wünsche
The University of Auckland,
New Zealand
burkhard@cs.auckland.ac.nz

Nathan Holmberg
77-Pieces Ltd,
New Zealand
nathan@77-pieces.com

ABSTRACT

Photorealistic rendering of fabric is essential in many applications ranging from movie special effects to e-commerce and fashion design. Existing techniques usually render the fabric's microscale structure. However, this can result in severe aliasing and is unsuitable for interactive cloth simulation and manipulation. In this paper we describe a novel real-time level-of-detail fabric rendering technique. The algorithm adjusts geometry and texture details with changing viewpoint by using a mipmapping approach, in order to obtain a perceptually consistent representation on the screen. Compared to previous work we also introduce more parameters allowing the simulation of a wider range of fabrics. Our evaluation demonstrates that the presented approach results in realistic renderings, increases the shader's run-time speed, and reduces aliasing artifacts by hiding the underlying yarn geometry.

Keywords: fabric rendering, anisotropic materials, real-time rendering, cloth simulation, anti-aliasing, level-of-detail methods

1 INTRODUCTION

Realistic fabric rendering addresses many different areas and industries in computer games and fashion applications. It is a challenging research field due to the complexity of the underlying fabric structure, textures, and materials, which results in complex light interactions and aliasing problems when using a raster representation. Fabric structures vary depending on the manufacturing process, such as weaving and knitting, and the desired fiber properties. Previous research in this field has explored different aspects of this problem, such as rendering complex weaving and knitting patterns, and developing specialized lighting models that simulate light interaction with the yarn geometry and its microstructure.

The modeling of complex weaving patterns and yarn geometry can result in aliasing when the screen resolution is lower than the perceived color variations on the material (caused by the geometry, lighting and texture). This is particularly problematic when animating the fabric using cloth simulations, which creates conspicuous temporal aliasing artifacts. In recent years, many hardware anti-aliasing techniques have been developed for real-time applications such as computer games, but are mainly used as a post-processing remedy. In this paper, we describe a level-of-detail fabric rendering technique for reducing the aliasing artifacts with minimal impact on computation time. This method can be used in conjunction with post-processing anti-aliasing techniques to further reduce the aliasing artifacts.

We want to create a parameterized fabric shaders for fashion design and e-commerce applications. This fundamentally requires fast interactive speed, high memory efficiency, and high robustness to support for a wide range of woven fabrics. We found that the model proposed by Kang [Kan10] would be the most suitable for our needs with several extensions and modifications to the original algorithm [YW11]. We adopted this model and implemented it in OpenGL Shading Language to support real-time parameterization of weaving pattern and yarn geometry.

Section 2 reviews existing fabric rendering techniques. Section 3 introduces previously presented techniques for modeling light interaction and rendering fabric, which form the foundation of our fabric rendering framework. Section 4 proposes our level-of-detail algorithm and improvements to the existing algorithm for real-time fabric rendering. Section 5 presents an evaluation of our framework and Section 6 draws conclusions and suggests directions for future work.

2 LITERATURE REVIEW

Fabric rendering techniques have been an active area of research since the early 1990s. We classify existing techniques into two categories, example-based and

procedural-based models. Example-based models focus on capturing reflectance information of specific material and use the captured data for rendering virtual fabrics. Procedural-based models are empirical mathematical models that use various parameters to control the appearance of fabrics.

2.1 Example-Based Models

Example-based cloth rendering techniques require the capturing of reflectance information of materials. This usually requires modification of the lightings, sensors, and planar examples of the corresponding materials. The reflectance properties of a material for different viewpoints and light directions can be encoded in a Bidirectional Reflectance Distribution Function (BRDF), which is often obtained with a gonioreflectometer [War92].

Daubert et al. [DLH01] proposed a Spatially-Varying Bi-directional Reflection Distribution Function (SVBRDF) specifically for cloth rendering using the Lafortune reflection model. It is designed to render clothes with repeating patterns such as knitting and weaving patterns. A texture map is used as a look-up table, for storing precomputed parameters of the Lafortune reflection model. This method works for both knitted and woven clothes by modeling the structure explicitly through the generation of new triangle meshes, but the computation consists of several rendering passes. Even though many methods have been proposed to obtain a SVBRDF using only a single view of a material sample [WZT+08], SVBRDF is generally very memory intensive, which makes it impractical for real-time applications where material parameters are interactively changed.

Another popular example-based model is the Bidirectional Texture Function (BTF), which captures a material's light interaction for varying light source and camera positions. The BTF captures more effects than the SVBRDF including self-shadowing, occlusion, and inter-reflection effects, and is used for many surface reflectance data measurements. The actual textures of the cloth samples are used and stored as texture maps, and they are used at render time with different parameters such as the illumination, camera position, and the texture coordinates of the object. Due to the higher number of parameters, the BTF suffers from high memory requirements and acquisition costs. Kautz [Kau05] introduced a compression method for the BTFs. He acquires a lower number of images and interpolates between them. Despite these compression approaches, example-based methods still require an over-abundant storage capacity for practical use, and they do not offer enough flexibility for rendering different types of clothes with different weaving patterns.

A volumetric approach that uses the microflake model was proposed by Zhao et al. [ZJMB11]. The approach acquires a volume model of the material that needs to be rendered using a X-ray computed tomography (CT) scanner. The volumetric data acquired is post-processed for orientation extraction and noise removal, and is matched to a photograph of the same material captured to obtain the optical properties for fabric rendering [ZJMB11]. This approach suffers from high memory requirements, due to the size of volumetric data, where each fabric sample takes approximately 7.26GB [ZJMB11]. It is also difficult to acquire equipments for this approach, due to the cost of CT scanners, thus making it difficult to capture different fabrics.

2.2 Procedural-Based Models

Procedural-based cloth rendering techniques are models that are designed based on the analysis of fabric structure. Yasuda et al. [YYTI92] developed shading models for woven cloth by analyzing fiber properties and weaving patterns. The authors proposed a tiny facet model for fabric materials taking into consideration reflection and refraction of multiple fabric layers. Using a multiple layer model, they simulated the scattering effects of fabrics by calculating the light refraction at different layers [YYTI92]. The reflection model assumes a simple percentage of warp and weft yarns in woven clothes and used a non yarn-based reflection. The light interaction with a small area of fabric is calculated by obtaining the total reflections [YYTI92]. Hence, this approach does not explicitly model the weaving patterns, but simulates the appearance of the fabric at a higher level where the weaving patterns are not visible.

Ashikhmin et al. [AS00] developed a microfacet-based anisotropic model that can be used for general materials, and was tested by simulating satin and velvet. The authors take into account the weaving pattern of satin and velvet. For example, satin is modeled by weighting the BRDF values of weft and warp yarns [AS00]. Due to a lack of self-shadowing and light interaction at the yarn-level, this microfacet anisotropic model is too generic to be used directly for fabric rendering, but it formed the foundation for many subsequently developed techniques.

Adabala et al. [AMTF03] use a weaving pattern input defined by the user, and generate a corresponding Anisotropic BRDF, texture, and horizon map for the clothing material. The authors render the fabric based on the weaving pattern input provided by the user to generate the overall appearance of the cloth [AMTF03]. This approach extends previous fabric models and allows more complicated weaving patterns to be defined. However, the authors applied the Ashikhmin-Shirley BRDF [AS00] on the object-level rather than the yarn-level, thus the modeling of light interaction with the fabric lacks realism compared to techniques which calculate light interaction based on yarn material and weaving patterns.

Kang [Kan10] proposed a procedural method that models the reflectance properties of woven fabric using alternating anisotropy and deformed microfacet distribution function. The proposed method is based on the microfacet distribution function (MDF) along with the Ashikhmin-Shirley [AS00] anisotropic shading model. Each sample point on the cloth is classified as a weft or warp yarn, and a corresponding distribution function is used accordingly to calculate the reflectance of that point [Kan10]. The alternating anisotropy approach allows the lighting to be defined for weft and warp thread by rotating the microfacet distribution function. Further deformation of the MDF enables the elaboration of yarn geometries and twisted appearances on the surface of each yarn. This approach enables not only the rendering of anisotropic clothes, but also the rendering of bumpy surfaces created by the weaving structure of the fabrics [Kan10].

Irawan [Ira08] developed a reflectance model and a texture model for rendering fabric suitable for distant and close-up views. The reflectance model is defined by the scattering effect of the fabric, while the texture model incorporates highlights calculated from the reflectance model. The texture model (BTF) is generated on the fly using parameters to control the types of yarn (i.e. staple or filament), and the appropriate weave pattern, and the yarn geometry is captured by using the fiber twisting angle to render the highlight on each yarn. A downside of this approach is the lack of shadowing and the masking effects to render some types of fabrics realistically, such as denim. However, the results look convincing and the approach is fully procedural, with intuitive variables at the fiber level, the yarn level, and the weaving pattern level.

3 FABRIC RENDERING MODEL

This section explains two techniques adopted by us in more detail: Kang's fabric rendering model [Kan10] and the Ashikhmin-Shirley anisotropic shading model [AS00] for capturing anisotropic reflections.

3.1 Ashikhmin-Shirley BRDF

The Ashikhmin-Shirley BRDF [AS00] is given by the following equation:

$$\rho(k_1,k_2) = \frac{p(h)P(k_1,k_2,h)F(k_1h)}{4(k_1n)(k_2n)(nh)} \qquad (1)$$

This equation represents the illumination of a point with the incoming light vector k_1 and outgoing light vector k_2 where additional functions explained below describe the effect of the microfacet structure. The vector n represents the surface normal at a point, and the vector h describes the half vector obtained from the incoming and outgoing light vector. The function $P(k_1,k_2,h)F((kh))$ captures the shadowing effects caused by microfacets. The function $F(kh)$ is the Fresnel reflectance that describes the amount of incoming light that is reflected off the surface specularly. The function $p(h)$ is the MDF given by Equation 2. It describes the illumination effects of weaving patterns in Kang's model [Kan10].

3.2 Microfacet Distribution Function

The microfacet distribution function characterizes a surface's distribution of microfacets, by encoding their normal direction relative to the underlying surface.

$$p(h) = \frac{\sqrt{(x+1)(y+1)}}{2\pi}(h\cdot n)^{xcos^2\phi+ysin^2\phi} \qquad (2)$$

The microfacet distribution function in Equation 2 is used to generate the BRDF. The function captures the visual effects of microgeometry, where the reflected specular light on the surface is proportional to the probability of the microfacet surface normals that are aligned to the half vector. Variables x and y controls the shape of the specular highlight and the intensity in anisotropic reflection.

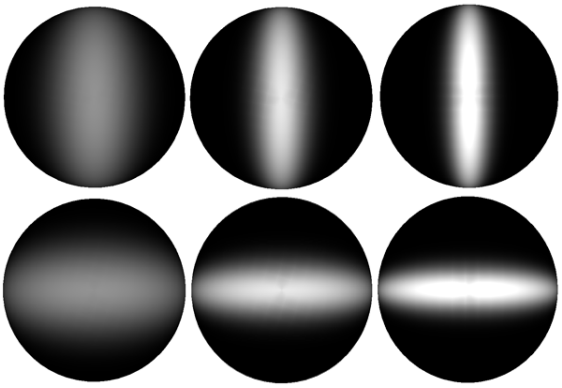


Figure 1: Our generated images using the Ashikhmin-Shirley BRDF [AS00] for visualizing the difference in specular highlights with varying parameters x and y. Top row: x = 10, 30, 50, while y stays constant equal to 1. Bottom row: y = 10, 30, 50, while x stays constant equal to 1.

A visualization of the microfacet distribution is shown in Figure 1. When the x and y values are close to each other, then the distribution of microfacets aligning to h is spread more evenly across the surface. The top row of the diagram demonstrates that if the x-value in the MDF increases from 10 to 50, then the distribution of microfacets becomes denser in the center, thus resulting in a less spread specular lobe on the object surface. This results in an increasingly narrow highlight stretched in y-direction.

3.3 Weaving Pattern Rendering

Kang [Kan10] proposed an alternating anisotropy solution to render weaving patterns by using Equation 2

to generate the specular highlight that can be seen on weaving pattern [Kan10]. Using the fact that weaving patterns are only made of weft and warp yarns, the specular highlight of weft yarns must therefore be obtained by rotating the microfacet distribution by 90°. This is again shown in Figure 1. The rotated microfacet distribution is seen on the second row, and is done by exchanging the values of x and y to create such rotation.

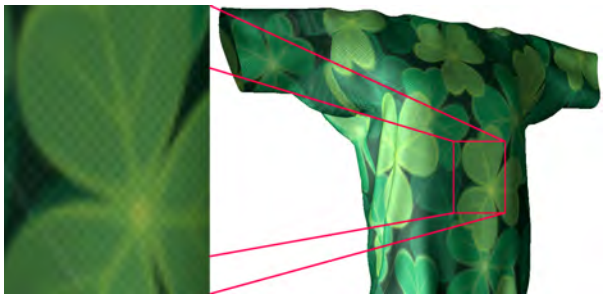


Figure 2: Weave pattern generated by using the alternating anisotropy method showing the closer view (left) and distant view (right).

An example of a weaving pattern generated using the alternating anisotropy method is shown in Figure 2. Without the yarn geometry the weaving pattern looks like a checkerboard pattern, thus the yarn geometry has to be defined for close-up viewpoints.

3.4 Yarn Geometry

The yarn geometry is generated after the weaving pattern by normal perturbation at run-time. Kang [Kan10] proposed that the yarn geometry can be specified by several parameters including: number of threads for each strand of yarn N_ξ, fiber curvature of each strand c_f, and the yarn curvature c_y. Therefore, the normal of each point is altered using these parameters and its sampling position on the weft or warp yarn [Kan10].

$$\text{weft:}(\delta x, \delta y) = (c_f(2\text{fract}(N_\xi(u_w - s_f\sigma^u)) - 1, c_y\sigma^v) \quad (3)$$

$$\text{warp:}(\delta x, \delta y) = (c_y\sigma^u, c_f(2\text{fract}(N_\xi(v_w - s_f\sigma^v)) - 1) \quad (4)$$

Equations 3 and 4 show the changes made to the x- and y-coordinates of the normal. The z-coordinate of the perturbed normal is generated by calculating $\sqrt{1.0 - \delta x - \delta y}$. This achieves yarn based lighting by taking into account different user-defined parameters including: fiber curvature (c_f), yarn curvature (c_y), slope of fibers (s_f), offsets in yarn space (σ^u and σ^v), and number of fiber strands (N_ξ) used to make up each yarn. The variables u_v and v_w are texture coordinates of the model, and the function fract() calculates the fraction component of a floating point.

Figure 3 shows a fabric model with weaving pattern and yarn geometry. The image on the left of Figure 3 shows a rendering of the fabric seen from a distance, with the microscale details well below an individual pixel size. The results are not significantly different to the version of rendering without yarn geometry as shown in Figure 2. The difference between the two fabric models becomes more visible when rendering a close-up view. The image on the right of Figure 3 illustrates that each yarn is constructed by several threads, as specified by the variable N_ξ, with a high yarn curvature resulting in large shaded areas on the sides of the yarns.

4 DESIGN

Fabric rendering techniques often suffer from strong aliasing effects if the resolution of the fabric microstructure is higher than the screen resolution it is displayed on. While post-processing can soften these effects, the solution is inefficient and artifacts can remain, especially for interactive applications such as cloth simulations. Some solutions use large weave sizes to reduce aliasing effects, but this is not suitable for fashion applications where realism is essential. We analyzed existing cloth rendering techniques and found that the method from Kang [Kan10] is the most promising one [YW11]. A major advantage of this algorithm is its speed, which was shown to be only 1.846 time more expensive than Gouraud shading [Kan10]. The approach also displays a high level of realism, as the results of rendered woven fabrics look realistic in close-up. However, this approach lacks a coloring scheme at the yarn level to render fabrics such as denim, and it also displays blatant aliasing artifacts on the fabric surface.

4.1 Level-of-detail Fabric Rendering

We propose a level-of-detail design for the fabric model proposed by Kang [Kan10], which removes unnecessary detail, and only renders the detailed fabric structure in close-up views.

Our design consists of two levels of visually perceivable fabric details: weave structure and yarn geometry. The visibility of yarn geometry is determined by the mipmap level, which is calculated with the use of derivative functions on texture coordinates. We explicitly render two mipmap levels for the general weave structure and the underlying yarn geometry. We limit the mipmap levels to between 0 and 1, and uses it as an alpha value for blending between the two layers of mipmap. This concept is shown in Figure 4, for those fragments that are highlighted in lighter colors, they are rendered with the general weave structure, whereas for those that are highlighted in darker colors, they are rendered with more detailed yarn geometry. This means that if the texture coordinates between neighboring fragments are changing quickly, then a higher level

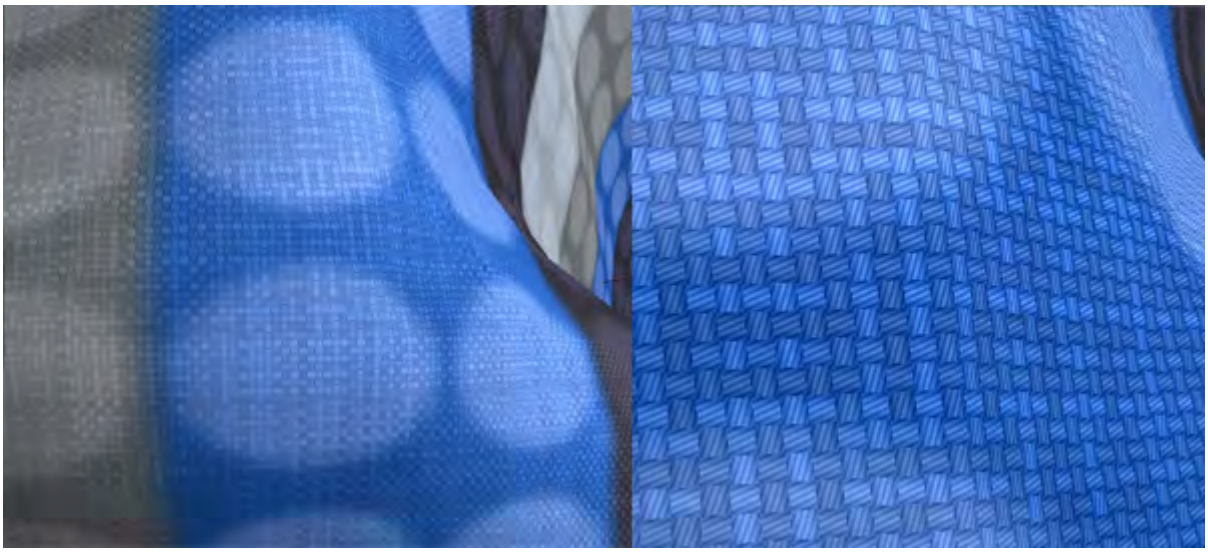


Figure 3: Rendering of weave pattern with yarn geometry seen from distance (left), and from close-up (right).

mipmap (less detail) is used, thus avoiding the rendering of unnecessary detail when the fabric is observed in a larger distance.

Figure 4: Visualization of mipmap level selection from far view point to close view point (left to right).

In essence, two MDFs are calculated, one using the pre-perturbed normals for weaving pattern rendering, and the other using the perturbed normals for yarn geometry rendering. The yarn geometry is obtained from the normal perturbation using Equations 3 and 4, which is then used as an input to the MDF. In practice, however, only the dot product between the halfway vector h and the normal vector n has to be recalculated two times for each values, with the rest of the calculations in Equation 2 only calculated once. Equation 5 shows the calculation of the final MDF, which is done by using the two previously mentioned MDFs, and weighting them with the α value obtained from the mipmap calculation that determines which level of mipmap should be used for rendering.

$$p(h) = (1.0 - \alpha)p(h_1) + \alpha p(h_2) \tag{5}$$

4.2 Extensions for Real-time applications

We also extend the model developed by Kang [Kan10] to support more types of fabrics.

4.2.1 Extended Fabric Coloring Scheme

For our system, we require the visualization of fabrics such as denim. Denim is often constructed from fiber using the twill-weaved weaving pattern. In contrast to ordinary cotton twill, denim uses different colors for the weft and warp yarn, with the most popular combination being white for the warp yarn and blue for the weft yarn.

We define extra parameters to specify the base color of individual weft and warp yarns, both in terms of diffuse and specular colors. Using these base colors, we apply Kang's algorithm [Kan10] to generate the procedural textures with weaving patterns and yarns to simulate the virtual fabrics.

4.2.2 Ambient Occlusion

The original method by Kang [Kan10] introduced an ambient occlusion term defined by the z value of the perturbed normal. Since the perturbed normal is generated to define the yarn geometry at the micro-level, the ambient occlusion term only works for scaling the anisotropic reflection at the yarn level to create a shadow effect on the reflection.

The self-shadowing effects at a higher level are not captured due to the lack of indirect lighting involved in calculating the overall reflectance of the fabric. However, self-shadowing is common in practice, e.g. when cloth is folded. Hence, we use Screen-Space Ambient Occlusion (SSAO) [Mit07] as a real-time ambient occlusion method to introduce self-shadowing effects to the existing fabric rendering method.

In the initial pass, we render our fabric at the same time as normal buffer and position buffer to avoid rendering the same object in multiple passes. We store fabric rendering results in a color buffer, and the color buffer is referred to for scaling its values using the calculated

ambient occlusion from the normal buffer and position buffer.

$$ao = max(0.0, \frac{dot(N,V)}{1.0+d}) \quad (6)$$

Equation 6 describes the calculation of the ambient occlusion of a pixel. A point (occluder) occludes another point (occludee) if the dot product between the normal of the ocludee and the vector from the ocludee to occluder is greater than zero, i.e if the point is located at the front face of the occludee, then it contributes some amount of occlusion scaled by the dot product and the distance between two points. In order to calculate the ambient occlusion at each pixel of the screen, neighboring pixels are sampled randomly using a noise function and the ambient occlusion value is averaged according to the number of sample points [Mit07].

4.2.3 Anti-Aliasing

The original implementation of the renderer produced clearly visible aliasing effects and moire patterns due to high frequency textures. These artifacts are caused by the microscopic details of the procedural textures generated in real-time. When the object is in motion, the aliasing artifacts become even more visible to temporal aliasing.

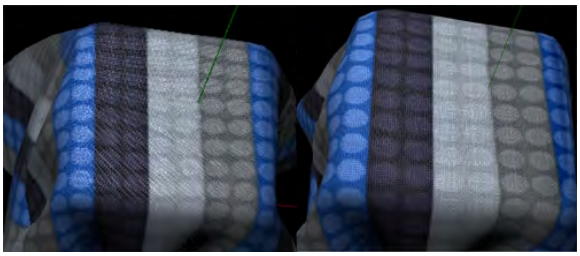


Figure 5: Aliasing of weave patterns. Comparison of fabric being viewed from a distance (left) and from close-up (right). The fabric was rendered with the original implementation by Kang [Kan10] without any anti-aliasing or level-of-detail algorithm

The left image of Figure 5 displays the distortion of weaving patterns when the viewpoint is located far away from the object. The moire patterns on the surface of the fabric are clearly visible as distorted curve lines. The fabric on the right in Figure 5 shows the weaving pattern when the viewpoint is at a closer distance to the object - no moire pattern is visible. Another example is given by Figure 6, which shows a denim fabric rendered without any underlying textures, but using blue colored weft yarns and white colored warp yarns. When the fabric is viewed from a distance (left image of Figure 6), the aliasing artifacts are more visible with this fabric due to the highly contrasted yarn colors between weft and warp yarns, also causing moire patterns to appear on the surface of the fabric. Furthermore, the twill-weave on the denim fabric is clearly distorted and unrecognizable from this distance. The aliasing artifacts are significantly reduced on the right of Figure 6, but still exist in high frequency areas such as regions where the fabric is bent around the underlying cuboid object.

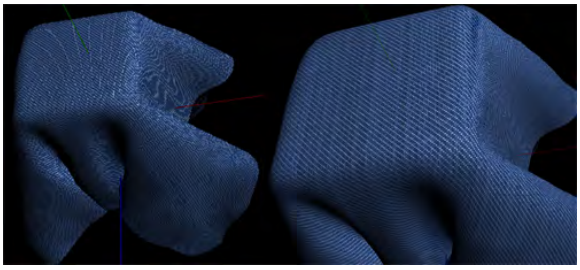


Figure 6: Aliasing of weave patterns of denim fabric. Comparison of distant viewpoint in magnified view (left) and close viewpoint (right)

An anti-aliasing method is required to reduce the moire effects on the surface of the fabric. While a level-of-detail scheme was proposed in Section 4.1, the size of the weaving patterns still introduces a high level of texture frequency on the fabric surface.

Traditionally, oversampling methods such as supersampling anti-aliasing (SSAA) and its variation, multisampling anti-aliasing (MSAA) were used to handle aliasing for graphics applications and computer games. SSAA is known to incur large bandwidth and shading cost, due to multiple numbers of samples being taken inside each pixel [HA90, Mam89], while MSAA improves on SSAA's performance by only evaluating each fragment value once and only supersampling depth and stencil values [Ake93].

Recently, post-processing anti-aliasing methods have become more popular for reducing aliasing artifacts. Methods such as morphological anti-aliasing (MLAA) [Res09] and fast approximation anti-aliasing (FXAA) [Lot09] have the advantage that they are independent to the rendering pipeline. The methods are applied at the last stage as an image-based post-processing technique. While MLAA resolves many edge aliasing problems, it is unable to handle pixel-sized features, and fails to reduce the moire-effects from high frequency textures [Res09]. FXAA employs a similar anti-aliasing procedure, where the image is used to detect edges using highly contrasting areas of each pixel, with an additional sub-pixel offset from the detected edge for low-pass filtering to achieve anti-aliasing in the sub-pixel level [Lot09]. Therefore, FXAA can be considered as a sub-pixel anti-aliasing technique and is hence potentially useful for our fabric shader. However, MSAA is the preferred choice due to its proven anti-aliasing performance over the entire object surface.

Despite the popularity of these methods, we found that they did not alleviate the high frequency aliasing prob-

lem we faced with texture aliasing from our implementations. Therefore, we decided to simply use an adaptive prefiltering approach [Ros05] inside our GLSL shader for averaging the pixel with its neighbors. The filter is adaptive such that the number of surrounding colors it calculates depends on the degree of similarity in each iteration. This algorithm is shown in Algorithm 1, which shows an overview of the algorithm for each fragment in calculating its final color. Essentially, the final color is iterated until its difference with other colors between neighboring fragments is less than a threshold, defined by the inverse distance of the fragment from the view point.

Algorithm 1 Prefiltering for fabric shader

```
count ← 1
ddx ← dFdx(worldPos) * 0.5
ddy ← dFdy(worldPos) * 0.5
while true do
    lastColor ← color
    color ← color + calcFragmentColor(worldPos
+ ddx * rand() + ddy * rand())
    count ← count + 1
    if count > 5 then
        δcolor ← lastColor − color
        if length(δcolor) < (1 / viewDistance) then
            break
        end if
    end if
end while
finalColor ← color/count
```

5 RESULTS

This section evaluates the effectiveness and efficiency of the improvements proposed by us. The following tests were performed:

- LOD fabric rendering quality test
- LOD fabric rendering performance test
- Denim Rendering

With rendering quality, we compare and contrast the quality of several images with real fabrics. To compare the effects of using level-of-detail rendering, we compare rendering results with and without the improvements, and we also compare their effects on aliasing artifacts. For rendering performance we compare the frame rates achieved with the different implementations. All tests were performed on a PC with Intel Core i7 2600k,12 GB memory at 1600 MHz with an AMD Radeon HD 6950 graphics card with 2GByte GPU memory.

5.1 Level-of-Detail Rendering

5.1.1 Rendering Quality

The level-of-detail rendering of woven fabric was tested by comparing the rendering quality of fabrics with and without our level-of-detail fabric rendering algorithm. Figure 7 shows that without level-of-detail rendering (left) many aliasing artifacts are seen, which they are not visible using level-of-detail rendering (right). The observations are confirmed by a second example shown in Figure 8, which represents a red twill-weaved woven fabric.

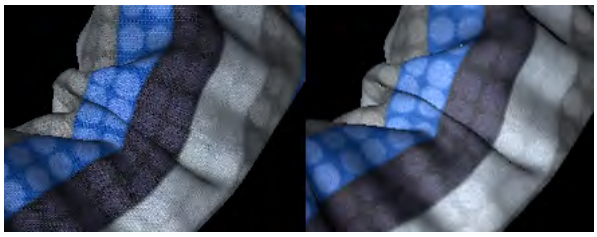


Figure 7: Level-of-detail rendering, without LOD (left) and with LOD (right).

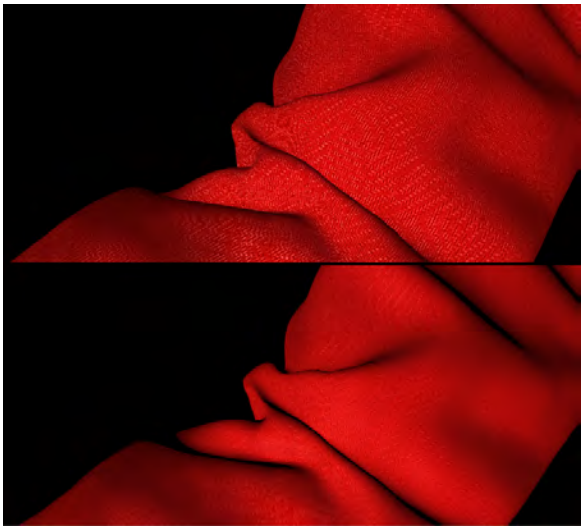


Figure 8: Level-of-detail rendering, without LOD (top) and with LOD (bottom).

The denim fabric still displays some aliasing artifacts with high frequency weaving pattern that is rendered on polygons facing away from the screen. This is due to the high contrast in color in the underlying weaving construct, coupled with the projection of weaving pattern to a much smaller projected area, thus making it difficult to smooth the high frequency changes in color at the micro-level. An example of this problem is shown in Figure 9, where the left image is a magnification of the small rectangle section in the right image. The left image illustrates some aliasing artifacts close to the edge of the fabric, with white lines going against the flow of the twill-weaved weaving pattern. These artifacts are not clearly noticeable in static images, but they

Implementation	Performance (ms)	Std. dev (ms)
Original	5.9302	0.0346
LOD	4.4037	0.04115

Table 1: Table comparing performance and standard deviation of the two algorithms in milliseconds per frame.

become very conspicuous in temporal aliasing when the fabric is in motion.

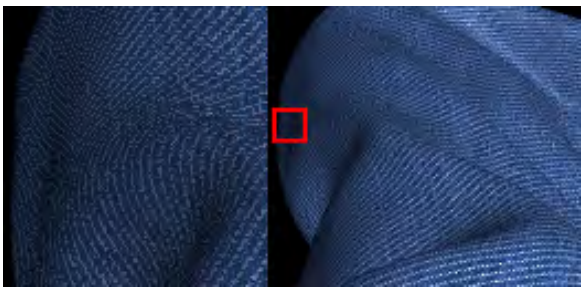


Figure 9: An example of aliasing problem due to texture projection. The left image is a magnification of the red square in the right image.

5.1.2 Performance Analysis

The performance of the two algorithms (the original algorithm and the LOD algorithm) was analyzed using an identical camera position and view, cloth model, texture, and input parameters for the rendering model. Both algorithms were tested along with the prefiltering approach, as we have decided to incorporate it to reduce the procedural texture aliasing.

Table 1 shows the results of the two algorithms, performed with a 3D model as shown in Figures 7 and 8, which contains 18892 vertices and 37566 triangles. Our level-of-detail fabric rendering algorithm is faster than the original algorithm. Given the improved rendering quality and reduced artifacts our new approach is hence preferable.

5.2 Denim Rendering

By extending the model to allow the specification of coloring of yarns, we managed to procedurally generate fabrics such as denim using the twill-weaved weaving pattern coupled with blue colored weft yarns and white colored warp yarns. Figure 10 provides a comparison between our results and a close-up of real worn denim fabric. The rendering results look realistic in terms of fabric structure, but more randomness and tear and wear needs to be incorporated into our model in the future.

We adopted the noise function proposed by Kang [Kan10]. The noise function generates random illumination at different yarns, which enhances the realism of our rendering. Note that some white warp yarns look brighter than others as is the case for real denim. So far our framework does not simulate washed denim fabric

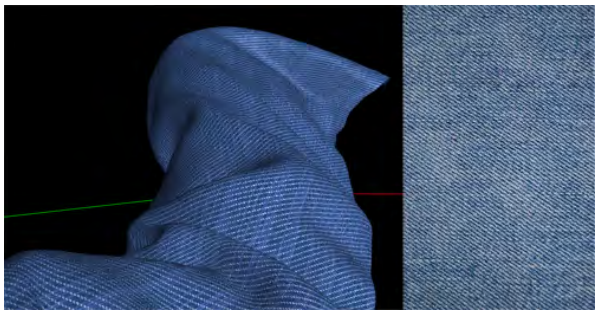


Figure 10: Denim fabric rendered with our framework (left) and a photo of real denim (right).

and hence we cannot replicate the random whitish patches in the image of real denim fabric.

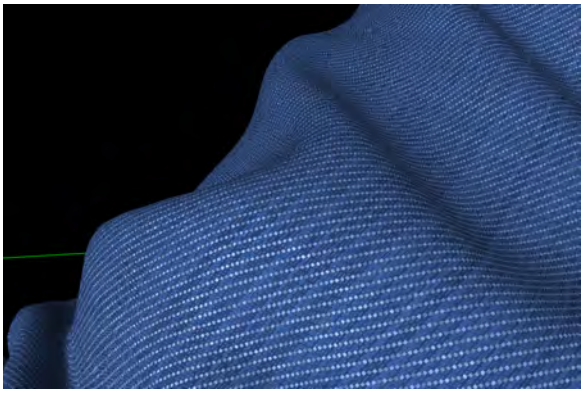


Figure 11: Close-up view of denim fabric rendered with our framework.

Figure 11 shows a close-up view of our rendering results. Our model renders the twill-weave too uniformly, lacking the natural imperfection found in real denim. When the denim is being viewed from a larger distance, the yarn geometry and weaving pattern gets aggregated and only the dominating color in the weaving pattern is visible to the user.

Figure 12 shows another close-up appearance of the denim fabric using a model of a pair of jeans.. The weaving pattern is defined to closely simulate the structure of a pair of real jeans. In order to render jeans realistically, we modified our fabric shader so that it supports the input of wash maps. Wash maps are used to define the patterns of washes for a pair of jeans, where washes are patterns of white patches on the jeans as shown in Figure 12. In Figure 12, a pair of rendered jeans is shown on the left with a pair of real jeans on the right. This close-up view demonstrates that the weaving pattern of the rendered jeans closely resembles the weaving pattern of the real jeans, as they are both similar in structure and size relative to the jeans model. Furthermore, at this viewing distance, the appearance of both jeans is very similar to each other.

A comparison of an entire pair of real jeans and our rendered jeans using a distance view is shown in Figure 13.

Figure 12: Jeans comparison when viewed from close-up: rendered jeans (left) and real jeans (right).

The rendered jeans (left) in Figure 13 closely resembles the real jeans (right) in Figure 13. From this distance, the weaving pattern is completely invisible to the observer, and aliasing artifacts are also unrecognizable on the fabric surface with the use of our LOD algorithm and prefiltering approach.

Figure 13: Jeans comparison when viewed from a distance: rendered jeans (left) and real jeans (right).

In our results, we found that that the use of direct lighting often makes the resulting rendered object too bright in areas that are occluded by the object itself. An example is shown in Figure 14, where the left image is the rendered jeans without ambient occlusion, and the right image is the rendered jeans with ambient occlusion. In this scene, the light is positioned to the left of the jeans, hence the inner thigh area should be dark as it is not directly illuminated by the light source. However, without ambient occlusion the rendered jeans still seems to be too bright around this area, and we found that the SSAO approach results in a more natural appearance of occluded areas.

Figure 14: Effect of ambient occlusion, before ambient occlusion (left), and after ambient occlusion (right)

6 CONCLUSION

In this paper, we analyzed several existing fabric rendering techniques. We chose to use the method proposed by Kang [Kan10] as a basis for our fabric shader, due to its rendering quality and performance. Several extensions were proposed to improve the robustness of the model and for supporting fabrics such as denim, and

ambient occlusion for enhancing the realism of self-occlusion of the cloth model. Furthermore, we proposed a level-of-detail approach in visualizing the aggregation of details with the use of a mipmap LOD selection mechanism, to help reduce aliasing artifacts resulting from high frequency textures. Overall, our extension to the model enabled us to successfully render denim fabric with an unwashed look and it significantly reduced aliasing problems. With the incorporation of wash maps to specify areas of washes, we have successfully replicated the overall and close-up appearance of actual jeans.

7 FUTURE WORK

The weave-based level-of-detail algorithm only reduces parts of the aliasing caused by high frequency textures. It still suffers from aliasing from small scaled weaving pattern and highly contrasting weft and warp yarns' colors, such as for denim fabric, depending on the size of weft and warp segments. Our approach rectified the aliasing problem that is often seen in weave-based fabric rendering approaches, but a better algorithm can be investigated in the future to reconstruct the appearance of high-detail level from lower levels, rather than filtering these details away.

8 REFERENCES

[Ake93] Kurt Akeley. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 109–116, New York, NY, USA, 1993. ACM.

[AMTF03] N. Adabala, N. Magnenat-Thalmann, and G. Fei. Real-time rendering of woven clothes. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 41–47. ACM, 2003.

[AS00] M. Ashikhmin and P. Shirley. An anisotropic phong BRDF model. *Journal of graphics tools*, 5(2):25–32, 2000.

[DLH01] K. Daubert, H.P.A. Lensch, and W. Heidrich. Efficient cloth modeling and rendering. In *Rendering techniques 2001: proceedings of the Eurographics workshop in London, United Kingdom, June 25-27, 2001*, page 63. Springer Verlag Wien, 2001.

[HA90] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. *SIGGRAPH Comput. Graph.*, 24(4):309–318, September 1990.

[Ira08] Piti Irawan. *Appearance of woven cloth*. PhD thesis, Ithaca, NY, USA, 2008. AAI3295837.

[Kan10] Y.M. Kang. Realtime rendering of realistic fabric with alternation of deformed anisotropy. *Motion in Games*, pages 301–312, 2010.

[Kau05] J. Kautz. Approximate bidirectional texture functions. *GPU Gems*, 2:177–187, 2005.

[Lot09] T. Lottes. FXAA. 2009. Also available as `http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf`.

[Mam89] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9(4):43–55, July 1989.

[Mit07] Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[Res09] A. Reshetov. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 109–116. ACM, 2009.

[Ros05] R.J. Rost. *OpenGL (R) shading language*. Addison-Wesley Professional, 2005.

[War92] G.J. Ward. Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics*, 26(2):265–272, 1992.

[WZT+08] J. Wang, S. Zhao, X. Tong, J. Snyder, and B. Guo. Modeling anisotropic surface reflectance with example-based microfacet synthesis. In *ACM SIGGRAPH 2008 papers*, pages 1–9. ACM, 2008.

[YW11] W. Yuen and B. Wünsche. An evaluation on woven cloth rendering techniques. In *Proceedings of the 26th International Image and Vision Computing New Zealand Conference (IVCNZ 2011)*, pages 7–12, Auckland, New Zealand, November 2011. Also available as `http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ2011_YuenWuensche.pdf`.

[YYTI92] T. Yasuda, S. Yokoi, J. Toriwaki, and K. Inagaki. A shading model for cloth objects. *Computer Graphics and Applications, IEEE*, 12(6):15–24, 1992.

[ZJMB11] S. Zhao, W. Jakob, S. Marschner, and K. Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph*, 30(44):1–44, 2011.

Error Metrics for Smart Image Refinement

Julian Amann
amannj@in.tum.de

Matthäus G. Chajdas
CG/Vis Group, CS Departement
Technische Universität München
Boltzmannstrasse 3
85748 Garching, Germany
chajdas@tum.de

Rüdiger Westermann
westermann@tum.de

ABSTRACT

Scanline rasterization is still the dominating approach in real-time rendering. For performance reasons, real-time ray tracing is only used in special applications. However, ray tracing computes better shadows, reflections, refractions, depth-of-field and various other visual effects, which are hard to achieve with a scanline rasterizer. A hybrid rendering approach benefits from the high performance of a rasterizer and the quality of a ray tracer. In this work, a GPU-based hybrid rasterization and ray tracing system that supports reflections, depth-of-field and shadows is introduced. The system estimates the quality improvement that a ray tracer could achieve in comparison to a rasterization based approach. Afterwards, regions of the rasterized image with a high estimated quality improvement index are refined by ray tracing.

Keywords

hybrid rendering, reflection error metric, depth-of-field error metric

1 INTRODUCTION

Nowadays, rasterization based graphic pipelines dominate real-time 3D computer graphics, because graphics hardware is highly optimized for this rendering algorithm. Many years of research have been spent on developing massive parallel processing units that are able to process complete pixel quads in a fast way by exploiting frame buffer locality and coherence [KMS10].

Even though rasterization has a lot of advantages, it also has some limitations. It performs well evaluating local illumination models, however there are problems with global effects like reflections. Because rasterization is limited to local illumination models, it is hard to compute physically correct reflections of the environment. For that reason, approximations like environment maps [Gre86] are used, which can result in visually plausible reflections.

Shadows are also a very challenging problem for rasterization. Although there are numerous shadow mapping and shadow volume techniques, they all have some inherent problems that arise from the local view of shading in the rasterization process. For example, most shadow mapping techniques suffer from the well-known biasing problems or have shadow mapping artifacts due to a too small shadow map resolution [ESAW11]. The number of annually appearing publications to shadow topics proves that the generation of shadows is still a challenging subject.

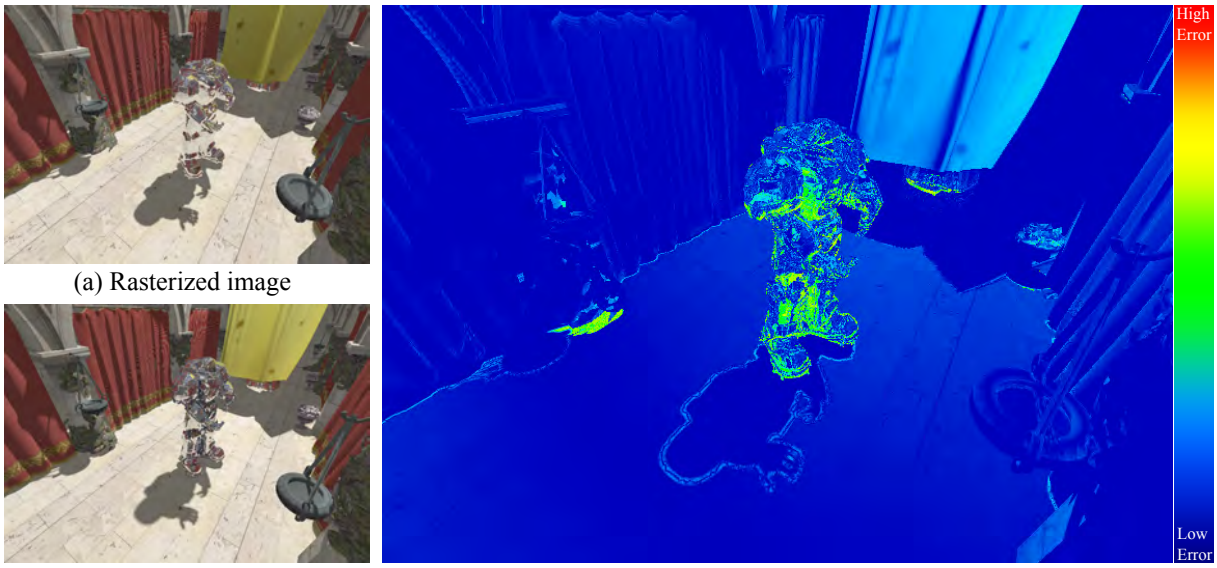
Besides reflections and shadows, it is also hard to simulate a correct thin-lens camera with rasterization. Most depth-of-field techniques which are rasterization based compute the circle of confusion and then just blur the image with the computed radius, which results in an incorrect depth-of-field effect [Pot81].

Secondary effects, like shadows or reflections are hard to achieve with a rasterization approach. A ray tracer can natively handle shadows and multiple reflections just by sending additional secondary rays. With a ray tracer it is not hard to simulate these effects. A thin-lens camera model can also be easily implemented in a ray tracer to get a nice depth-of-field effect.

2 MOTIVATION

Because ray tracing is computationally very expensive, rasterization is still used for games today. Another reason is that under some circumstances a rasterizer can produce exactly the same image as a ray tracer could. Figure 1 compares a scene rendered with ray tracing to a scene rendered with rasterization.

Remarkable is the fact that the rasterized image took about 7 ms to render with precomputed environment maps on commodity hardware (NVIDIA GeForce GTX



(a) Rasterized image

(b) Ray traced image

(c) Difference image

Figure 1: A difference image of scene (a) that has been rendered with rasterization (b) and ray tracing (c). The temperature scale on the left shows how to interpret the colors of the difference image. Blue colors mean a low difference and red colors mean a high difference. For example high differences can be seen in the reflecting object in the center of scene.

460) at a resolution of 762 x 538 pixels with no scene graph optimizations. However, the ray traced image with the same image quality and resolution and a highly optimized scene graph took roughly 976 ms to render (the performance measurements were made with the render system described in section 7).

Being given the choice of those techniques, one has to weigh the speed up against the high image quality. In order to profit from the advantages of rasterization and ray tracing, a hybrid approach is preferable. For example, a hybrid technique can just use ray tracing to compute reflections while the rest of the scene can be conservatively rasterized. This helps to get high quality reflections at the cost of only ray tracing the parts of the scene that have reflecting materials.

Ideally, one would estimate beforehand how big the difference between the rasterized and ray traced image is. This difference (see Figure 1c) could be used as a hint to find out where it is most appropriate to send some rays to improve the image quality.

3 CONTRIBUTIONS

This paper describes an error metric for reflections, shadow mapping and depth-of-field, which estimates the expected pixel error between the rasterized and the ray traced scene. The error metric is a heuristic one, which yields an expected but not an absolutely correct error value. During the rasterization of the scene, the error value is calculated and stored in an additional render target. After a first approximation of the current scene by rasterization, the error value of each pixel can be used to find out where it is most appropriate to refine the rasterized image by ray tracing.

This paper also presents a scheduling strategy that determines in which order the parts of the image are getting refined via ray tracing, so the scene converges quickly. This means that parts of the scene with high error get ray traced first and more often than parts of the scene with a low error. This helps to prevent wasting a lot of computation time on parts of the image where no difference or only a slight difference between the ray traced and the rasterized version is observable.

Furthermore, the implementation of a hybrid rasterization and ray tracing framework that is based on Microsoft Direct3D 11, NVIDIA CUDA Toolkit and NVIDIA OptiX 2.5[1] ray tracing engine is described, which demonstrates the feasibility of the described smart image refinement system. The system is called smart, because it refines the image according to the error estimate.

4 RELATED WORK

In the following section related work is briefly presented. Contributions made by other researchers reach from simple hybrid rendering systems to sophisticated perceptually-based techniques.

[1] NVIDIA OptiX is a freely available low level ray tracing engine that runs entirely on the GPU. Currently OpitX is only supported by NVIDIA GPUs. Similar as Direct3D or OpenGL provides an interface to an abstract rasterizer which can be used to implement various rasterization-based algorithms, OptiX provides an interface to an abstract ray tracer.

Perceptually-based techniques try to shortcut the render process by computing a perceptually indistinguishable solution instead of a fully converged one. In [YPG01], a perceptually-based technique is described that calculates a spatio–temporal error tolerance map. The computation of the error map takes a few seconds and is targeted at offline renderers. Each pixel of the error map indicates how much effort should be spent on the respective pixel. The error value is determined by harnessing knowledge about the human visual system and a model which predicts visual attention. For example, the eye is less sensitive to areas with high spatial frequency patterns or movements. Alongside with a prediction on where the viewer directs his or her attention, an estimate is computed that describes how important a pixel will be. This estimate is saved in the error map and is used during the render process to spend more time on important regions of the image. The paper's authors achieved on their test rendering system a $6\times$ to $8\times$ speedup.

[Cab10] uses a simple error metric. The metric consists of only a binary decision if ray tracing or rasterization is to be used. If the rasterizer renders a triangle with a transparent or reflecting material, a flag is set in a ray casting buffer. Afterwards all pixels marked with the flag get ray traced and combined with the rasterized image. They use a CPU-based ray tracer.

In [KBM10], a hybrid approach is shown that combines shadow mapping and ray tracing to render shadows. In a direct render pass and from a small number of shadow maps that are used to approximate an area light source by several point lights, a shadow refinement mask is derived. The mask is used to identify the penumbra region of an area light source. A pixel is classified as inside the penumbra when it cannot be seen from all point lights. Afterwards, the penumbra pixels are dilated by a 5×5 structuring element. The dilated region is then rendered by a CPU-based ray tracer to compute accurate shadows.

5 ERROR METRICS

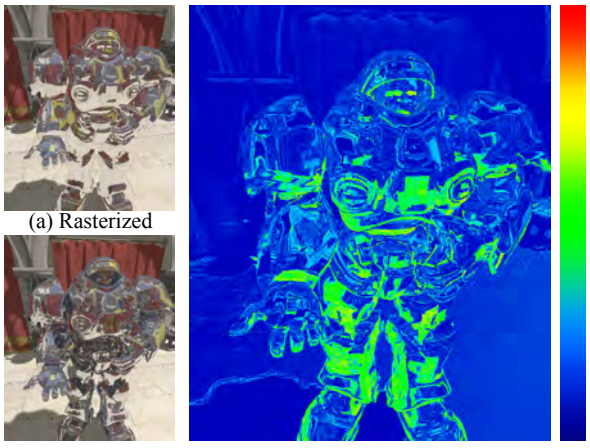
This section describes different error metrics for reflections, depth-of-field and soft shadows. The presented error metrics are used by the smart image refinement system to find out in which regions refinement by ray tracing is most appropriate. An error value is estimated for each pixel and is stored in an additional render target. The error metric is used as a heuristic that indicates how likely the calculated pixel is wrong in comparison to a pure ray traced version of the scene.

A high error value indicates that the approximation by the rasterizer contains a high error, whereas a small error value indicates low errors in the approximation by the rasterizer. The error value is just based on heuristics, which means that in certain circumstances, a high error value refers to a pixel that has only a small real or no approximation error at all compared to the ray traced scene. Conservative error metrics were chosen, so no pixels get estimated as correctly approximated, even if they are not correct.

Each error metric is normalized, which means it generates an error value in the range of $[0,1]$.

Reflections

Reflections can be approximated by environment maps in a rasterization based environment. Figure 2 compares reflections rendered by rasterization to reflections rendered by a recursive ray tracer.



(a) Rasterized (b) Ray traced (c) Difference

Figure 2: The rasterized image (a) approximates reflection with an environment of the scene. Figure (b) shows the same scene rendered with a recursive ray tracer. The difference image (c) visualizes the difference between Figure a and b. A red value indicates a high difference and a blue value a small difference.

As can be seen from the difference image, the approximated rasterized image is far from perfect. It contains several regions with wrong reflections.

The simplest to think of error heuristic is one that just makes a binary decision, depending on the criterion if a fragment is part of a reflection object or not [Cab10]. Assuming it is part of the reflecting material, the error metric returns 1, in all other cases it returns 0:

$$E_{reflection_1} = \begin{cases} 1 & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

The previous classification is a very simple one. A more sophisticated error metric can be derived, if we try to understand why the approximation of an environment is wrong. Figure 3 shows two major problems of environment mapping.

Put the case that we want to rasterize a reflecting sphere with the help of environment mapping. For a point P

on the sphere (see Figure 3) we need to look up the current reflection from the environment map. For the look up in the environment texture, we first need to compute the reflection vector. The look up vector is then transformed to texture coordinates which are afterwards used to access the reflection map. The problem with this approach is that the environment map has usually been rendered from the center of the reflecting object. This means, we get the color that is seen from the center of the environment map towards the direction of the reflected vector. Instead of this color, the correct color would be the color that can be seen from the point P towards the direction of the reflected vector. Figure 3 illustrates this. From point P, the reflection direction points toward the colored sphere (r_2). So we expect to see a colored sphere in the reflection of P. But in fact, the environment map technique uses the center of the environment map to look up the color of the reflected object. Looking from the center of the environment map into the direction of the reflection vector, a yellow cube can be seen instead of a colored sphere.

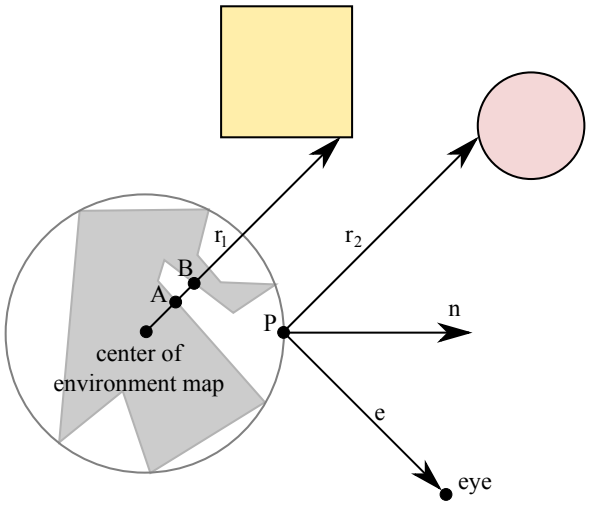


Figure 3: Environment mapping has two major problems: First of all each reflection is computed as if the reflecting point (P) would lie in the center of the environment map. Also it cannot handle self-reflections, which leads to incorrect shading results at point A.

The environment map technique would result in a correct shading, if the shaded point is located directly in the environment map center. For points that only have a very small distance to the environment map center, this approximation works as well. But for points with a further distance to the environment map center, the approximation by an environment map gets worse.

From this observation, a simple heuristic can be derived: The further a point is away from the environment map center, the more likely an environment map technique results in an incorrect approximation. This means that the distance between a point (p) of a reflecting object and the environment map center (c) needs to incorporate in the approximating environment map error metric:

$$E_{reflection_2} = \begin{cases} \frac{distance(p,c)}{maxDistance} & \text{: reflecting material} \\ 0 & \text{: else} \end{cases}$$

Another error metric can be deduced from the incident vector and reflection vector, as Figure 4 illustrates. Assuming that there is a scene with a reflecting sphere where the center of the environment map has been placed at the center of the sphere, this would mean that the environment map has been rendered from the center of the sphere. Looking at the reflecting sphere in a way that the incident ray (the ray from the eye point to a point in the scene) is hitting directly the center of the sphere, as this is the case for the incident vector I_1, the look up in the environment map will yield the correct reflection color.

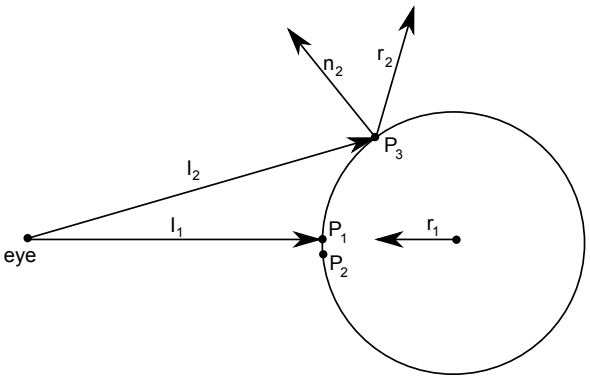


Figure 4: Incident and corresponding reflected vectors

The returned reflection color is correct because the given incident vector I_1 leads to the reflection vector r_1, which means we want to know what can be seen from the intersection point P_1 into the direction r_1. But this is exactly the same question as what can be seen from the center of the environment map into the direction of r_1. If we look from the eye point into the direction of a point that is near to P_1 like point P_2, the incident vector narrowly misses a hit with the center of the environment map, but the looked up direction in the corresponding environment map is approximately not too far from being correct.

It seems that for small angles between the incident and the reflection vector, the approximated reflection vector is almost correct, but for bigger angles like the angle between incident vector I_2 and r_2 it gets worse. From this property, the following error heuristic can be derived:

$$E_{reflection_3} = \begin{cases} \langle -i, r \rangle & \text{: reflecting material} \\ 0 & \text{: else} \end{cases}$$

It is assumed the reflection vector r and the incident vector (vector from eye point to shaded point) i in the

above equation are normalized. The angle between the vector r and $-i$ is always in the range $[0°, 90°]$. Since the angle can be in the range $[0°, 180°)$ the dot product fails for greater than 90° angles. To circumvent this problem instead of considering the reflected vector the normal can be considered which leads to the following equation:

$$E_{reflection_4} = \begin{cases} \langle -i, n \rangle & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

This works because the angle between the incident and reflected vector is directly proportional to angle between the reflected and the incident vector. The angle between the negative incident vector and the normal can never exceed 90°.

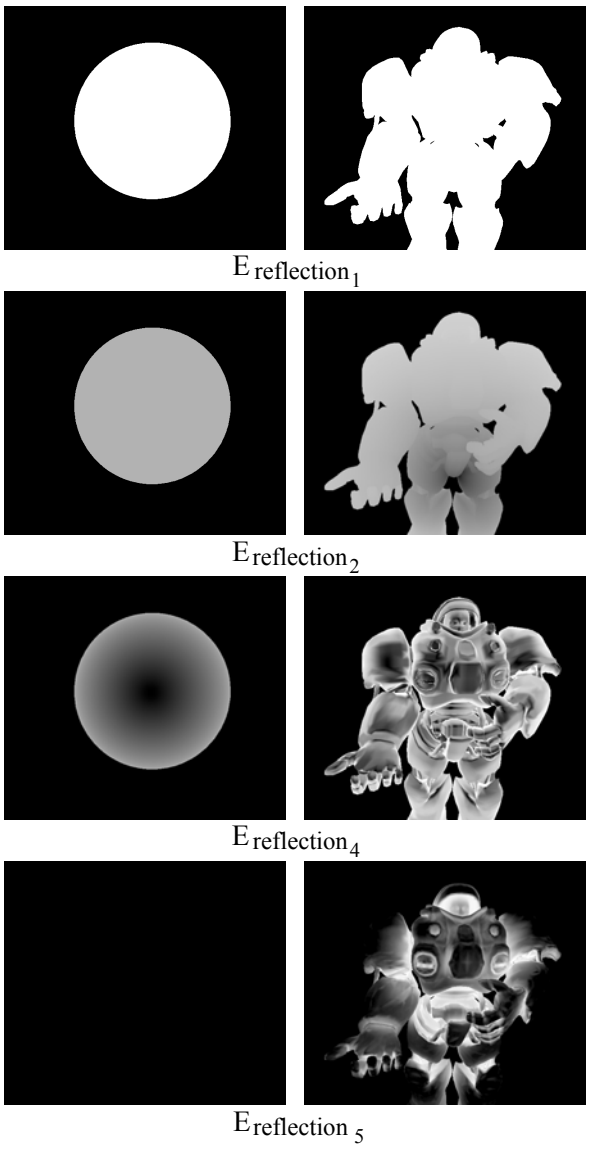


Figure 5: Displays the different reflection error metrics applied to a scene with sphere (left) and a scene with a more complex shape (right)

Another not yet considered problem of the error metric that is also related with environment maps are self-reflections. Concave objects are displayed incorrectly by an environment map technique. Figure 3 shows the reason for this. Assuming we want to compute the reflection of the point A in Figure 3 given the reflection vector r_1. In a look up in the environment, the yellow color from the yellow cube is returned. However in fact the reflection ray intersects the reflecting object itself (a so-called self-reflection) in point B and despite of this, the yellow color from the environment map is nonsense. Self-reflections can probably not be handled by environment maps. We can take care of this in our error metric by using an ambient occlusion map. The ambient occlusion map can be interpreted as a description of the curvature of the reflecting object. This information can be directly used in a heuristic that estimates the possibility of self-reflections:

$$E_{reflection_5} = \begin{cases} k_a(p) & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

$k_a(p)$ refers here to the ambient occlusion term.

Figure 5 shows the different error metrics applied to two sample scenes.

Depth-of-Field

Most rasterization based depth-of-field techniques are image based and use only the depth buffer and color buffer to compute a depth-of-field effect. Thereby, the information about the scene is lost. In a ray tracer, the lens is sampled at different locations. Each sample on the lens has a different view of the current scene. In the rasterizer approach, we have only one view at the scene from the center of the lens. This can lead to missing objects, because in some cases from some points on the lens, objects can be seen that cannot be seen from the center of the lens.

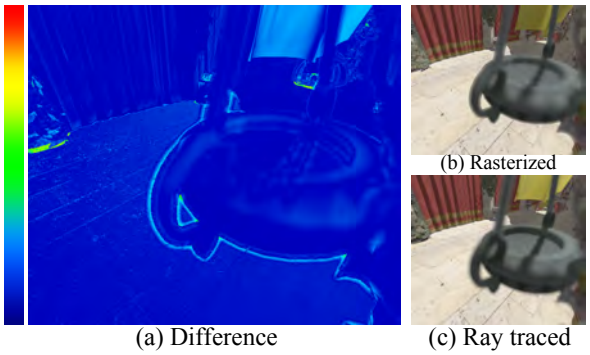
Another problem of most depth-of-field techniques is color leaking. Color leaking can be seen around sharp edges that are in focus in which other blurry objects from the background bleed into [LU08]. The other way around, objects in the foreground can bleed into objects in the background. Figure 6 shows this effect.

As demonstrated in Figure 7, rasterization based depth-of-field have problems in regions with high depth discontinuities. This knowledge can be exploited to construct an error metric for the depth-of-field.

To find depth discontinuities, an edge filter, like the Sobel filter, can be applied. A problem with this approach is that the founded edges have to be dilated by a structuring element, since the artifacts do not only occur at the identified edge pixels, but also in the neighborhood of the edge pixel according the circle of confusion. The

(a) Rasterized (b) Raytraced

Figure 6: A blurry foreground object bleeds into the focused background object.



(a) Difference (b) Rasterized (c) Ray traced

Figure 7: Difference image of scene (with applied depth of field effect) (a) that has been rendered with rasterization (b) and ray tracing (c). Regions with high depth discontinuities are problematic for rasterization based rendering techniques.

maximal radius of the circle of confusion C_{max} for the dilation can be determined for a point p by the following equation with image distance V_P, focal length F and aperture number n (z_p is the distance between the point p and the image plane):

$$C_{max}(p) = max(C(z_p), C_\infty)$$

$$C_\infty = \lim_{z \to \infty} C(z) = |F - V_p| \frac{1}{n}$$

For simplicity reasons, we use a quad shape structuring element in our implementation to approximate the circle of confusion. Figure 8 shows the error metric for depth-of-field.

The Error metric for depth of field can be expressed as:

$$E_{dof} = \begin{cases} 1 & : \text{Vicinity of a depth discontinuity} \\ 0 & : \text{else} \end{cases}$$

The described error metric is not absolutely conservative, which means that errors can also occur in regions

(a) Rasterized (b) Dilated

Figure 8: After the depth discontinues have been marked, they need to be dilated according to the circle of confusion.

that were not classified with an error value of 0. However, it can give a smart image refinement system a good hint where to start the refinement process.

Shadows

In [GBP06], an algorithm has been described that can be modified to estimate where the penumbra of a given area source light will be projected onto the scene. In Figure 9, the estimated penumbra region is shown in green color.

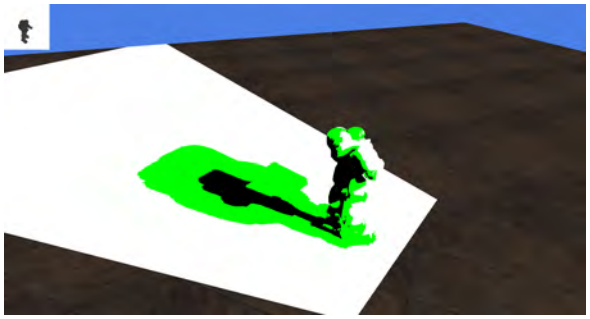


Figure 9: The estimated penumbra region is shown in green color.

The algorithm described in [GBP06] makes a conservative estimation of the penumbra region, and is therefore perfectly appropriate to be used as an error metric. The difference between the shadow generated by the rasterization system and the ray tracer is only notable in the penumbra region. In umbra regions and in regions where the area light source is not occluded by other objects (so that the scene is fully lit by the light source) no difference between the rasterizer and ray tracer is noticeable (see Figure 1 - only the penumbra region needs to be refined).

In [Mic07], an improved algorithm is presented which can estimate a tighter, conservative penumbra estimation than the algorithm described by [GBP06]. Even though it requires more memory, the tighter estimation reduces the amount of pixels that have to be refined, resulting in an overall improved performance.

The corresponding error metric for soft shadow is therefore quite simple:

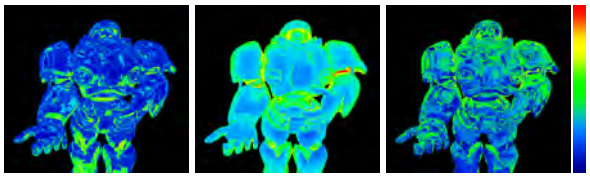
$$E_{shadow} = \begin{cases} 1 & : \text{Pixel resides in penumbra region} \\ 0 & : \text{else} \end{cases}$$

Combination of Error Metrics

The different error metrics can be combined in multiple ways. A naive idea is to calculate an averaged sum:

$$E_{combined_1} = \frac{\sum_{i=1}^{n} E_i(p)}{n}$$

Figure 10 shows the quality of the average sum metric.



(a) (b) (c)

Figure 10: Quality of averaged sum metric. (a) shows the real error, (b) the estimated error and (c) the difference image.

For a better estimation, a more complex combination is required. A simple, yet effective approach is to use a linear combination of the different error metrics (i.e. $E_{reflection_i}$, E_{dof}, E_{shadow}) and let the smart image refinement system automatically determine the coefficients λ_i by rendering multiple views of a scene with the goal of minimizing the difference between the estimated and the real error:

$$E_{combined_2} = \sum_{i=1}^{n} \lambda_i E_i(p)$$

The determination of the factors λ_i is done as a pre-process for each different scene. In this pre-process a certain number of screenshots from the scene is taken (with pure rasterization and pure ray tracing). Then a random tuple of λ_i coefficients is chosen and the combined metric $E_{combined_2}$ is then compared with the real error. We repeat this step multiple times and choose the λ_i coefficients which result in the best approximation for all test images.

6 SCHEDULING STRATEGY

This section describes how the error value is used to direct the rendering process of the smart image refinement system.

First the scene is rasterized by the rasterization system. During rasterization, an error value is also computed as described in the previous section about error metrics. After the rasterization pass, the color buffer and the error color buffer are filled. Now post-processing effects are applied to the previously rendered scenery. The post-processing result is written to the post-process color buffer; after this, the post-process error buffer is computed. Then the error buffer and the post-process error buffer get composed in a combined error buffer. For each pixel, an error value is computed and stored in the combined error buffer. After composing the error buffers, the next step is to sort the pixels. The error buffer also stores, besides the error value, the position for each pixel. The position is needed to find out to which pixel a corresponding error value belongs to after reordering them according their error values. After sorting the error pixels, they are gathered in the request buffer. Additionally to the position, a sample count value is also stored in the request buffer for each pixel that determines how many rays should be traced for the corresponding pixel. The sample count is determined by an estimation pass that fills the request buffer. After the request buffer is filled, it is handed over to the ray tracing system. The ray tracing system reads the first entry from the request buffer and samples the corresponding pixel according to the sample count. The ray tracer proceeds this process for a user-defined maximum number of pixels. After the maximum number is reached, the ray tracing process stops and the smart image refinement system continues with blending the current ray traced image with the computed rasterized image. The blending factor of each pixel depends on the total number of samples that were computed for the corresponding pixel by this time. Figure 11 gives an overview of this process. This process is repeated until the whole image is refined.

7 IMPLEMENTATION

For the implementation of the smart image refinement system Direct3D 11, CUDA 4.1 and OptiX 2.5 ([Ste10]) have been used. Direct3D is used for the rasterization part and analogously OptiX is used for the ray tracing part. Thus all rendering is done on the GPU. Optionally the system can perform pure rasterization with Direct3D or pure ray tracing with OptiX. In the case of pure ray tracing Direct3D is needed only to show the generated OptiX output buffer. Pure ray tracing and rasterization is used for comparison purposes like the time measurements in section 2 or in table 1.

The ray tracing subsystem uses a SBVH acceleration structure [SFD09] provided by OptiX to accelerate ray-triangle intersections. The rasterizer subsystem renders without any scene graph optimizations in a brute force manner.

A pixel shader is used to write the error values during rasterization to an additional render target (the error buffer). Some error values can be only determined after post-processing so there is an additional error buffer (post-process error buffer) which stores the error values determined during applying post-processing effects like depth-of-field. The combined error buffer which contains the unsorted error values is shared with CUDA.

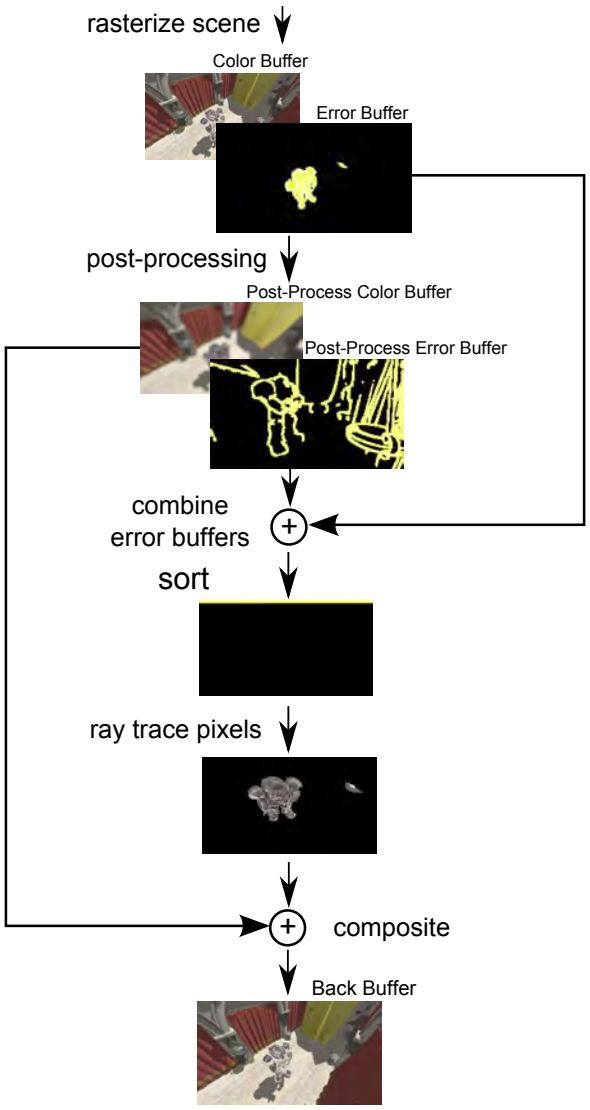


Figure 11: Overview of the smart image refinement system. During rasterization an error buffer is generated. The error buffer is sorted to give the ray tracing subsystem information where refinement makes most sense.

Thrust [HB10], a CUDA library is then used to sort all error values. The error values are encoded in such a way that the radix sort algorithm of Thrust can be used. After sorting the pixel values according their error values a full screen quad is drawn with the size of the sorted error buffer. In this step all pixel are gathered in a request buffer which is implemented as an structured buffer. The request buffer is list with the pixels that need to be refined. Since the error buffer has been sorted the list is also sorted according to the error value.

8 RESULTS

Table 1 shows a performance comparison of pure ray tracing and the prototypically implemented smart image refinement system.

Resolution	PRT in ms	SIR in ms	Error pixel (%)
800×600	440	406	188023 (39)
800×600	312	230	100934 (21)
800×600	203	79	20158 (0.4)
800×600	145	45	4181 (0.01)
1024×768	640	587	290542 (36)
1024×768	305	185	88063 (11)
1024×768	238	84	32889 (4)
1024×768	201	52	32889 (1)
1920×1080	1510	1463	805368 (39)
1920×1080	1107	901	499369 (24)
1920×1080	639	243	145239 (7)
1920×1080	484	113	44140 (2)

Table 1: Performance comparison of pure ray tracing (PRT) and smart image refinement (SIR). In the SIR implementation, one primary ray is traced for each error pixel. Also a shadow and reflection ray is cast per intersection. This is done recursively for reflections three times.

As can be seen from Table 1 smart image refinement is faster than pure ray tracing and at the same time it has the same image quality, provided that conservative error metrics are used. All measurements in this section were made with a NVIDIA GeForce GTX 560 Ti. As a test scene, the extended Atrium Sponza Palace scene that was originally created by Marko Dabrovic and extended by Frank Meinl has been chosen.

The sorting only has to be performed when the scene or camera orientation/position changes. In the implementation of the smart image refinement system a user-defined number of rays are always traced. For instance, the tracing of 8192 rays and the composition of the ray traced and rasterized image takes about 30 ms, depending on the current scene on camera view. This makes it possible to show the user first results after a short render time. Something that has be taken into consideration as well is the fact that in a pure ray traced based approach, the same number of samples is computed for each pixel, no matter if refinement makes sense for the corresponding pixel.

The performance of the smart image refinement system drops with higher error pixel rates. The major reason for this is that resources (e.g. error buffer, request buffer) have to be copied between Direct3D 11, CUDA and OptiX because they cannot be directly shared (some API extensions to improve the interoperability between OptiX, CUDA and Direct3D could avoid these copying steps). For example to hand over the error pixels that should be refined by OptiX, a request buffer has to be filled with the sorted data from a CUDA buffer. The CUDA buffer cannot be directly accessed by OptiX. The data has to be copied first.

9 CONCLUSIONS AND FUTURE WORK

In this work, a GPU-based hybrid rasterization and ray tracing system was presented that is able to direct the render processes to regions with high relevance. Regions with a high error are getting refined first and more often than regions with a small error value. This helps to converge fast to a stable image and avoids at the same time the waste of computing time in regions that do not need any refinement.

There is some scope for improvements of the described error metrics.

Besides reflections, shadows and depth-of-field, it would also be interesting to see how other effects like ambient occlusion (AO) or refractions can be integrated into a smart image refinement system. In the case of AO, a screen based ambient occlusion technique can be employed in the rasterizer to compute a fast approximation of an occlusion term.

Another interesting aspect that has not been considered in this work is global illumination. Global illumination could be approximated with light propagation volumes and refined with a more sophisticated ray tracing technique like path tracing.

There are several real-time perceptually based approaches like [CKC03] which try to cut down rendering time by focusing on important parts. These ideas can be combined with our approach.

10 REFERENCES

[Cab10] Cabeleira Joăo. Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time. http://www.voltaico.net/files/article.pdf, 2010.

[CKC03] Cater, K., Chalmers, A., and Ward, G. Detail to attention: exploiting visual tasks for selective rendering. Proceedings of the 14th Eurographics workshop on Rendering, Eurographics Association, 270-280, 2003.

[ESAW11] Eisemann, E.; Schwarz, M.; Assarsson, U. & Wimmer, M., Real-Time Shadows A. K. Peters, Ltd., 2011.

[GBP06] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In Eurographics Symposium on Rendering (EGSR), Nicosia, Cyprus, 26/06/2006-28/06/2006, pages 227-234. Eurographics, 2006.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. IEEE Comput. Graph. Appl., 6:21-29, November 1986.

[HB10] Jared Hoberock and Nathan Bell. Thrust: A parallel template library, Version 1.3.0. http://www.meganewtons.com/, 2010.

[KBM10] Erik Knauer, Jakob Bärz, and Stefan Müller. A hybrid approach to interactive global illumination and soft shadows. Vis. Comput., 26(6-8):565-574, 2010.

[KMS10] Jan Kautz Kenny Mitchell, Christian Oberholzer and Peter-Pike Sloan. Bridging Ray and Raster Processing on GPUs. High-Performance Graphics 2010 Poster, 2010.

[LU08] Per Lönroth and Mattias Unger. Advanced Real-time Post-Processing using GPGPU techniques, Technical Report, No. 2008-06-11, Linköping University, 2008.

[Mic07] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. Computer Graphics Forum, Vol. 26, No. 3, pages 515-524, 2007.

[Pot81] Potmesil, Michael and Chakravarty, Indranil. A lens and aperture camera model for synthetic image generation. In SIGGRAPH 81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques, pages 297-305, New York, NY, USA, 1981. ACM.

[SFD09] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In Proceedings of the Conference on High Performance Graphics 2009, HPG 09, pages 7-13, New York, NY, USA, 2009. ACM.

[Ste10] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. ACM Transactions on Graphics, August 2010.

[YPG01] Hector Yee, Sumanita Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. ACM Trans. Graph., 20:39-65, January 2001.

Multi-View Random Fields and Street-Side Imagery

Michal Recky
ICG TUGraz
Graz University of Technology
Inffeldgasse 16
A-8010 Graz, Austria
recky@icg.tugraz.at

Franz Leberl
ICG TUGraz
Graz University of Technology
Inffeldgasse 16
A-8010 Graz, Austria
leberl@icg.tugraz.at

Andrej Ferko
FMFI UK
Mlynská dolina
842 48 Bratislava, Slovakia
ferko@sccg.sk

ABSTRACT

In this paper, we present a method that introduces graphical models into a multi-view scenario. We focus on a popular Random Fields concept that many researchers use to describe context in a single image and introduce a new model that can transfer context directly between matched images – Multi-View Random Fields. This method allows sharing not only visual information between images, but also contextual information for the purpose of object recognition and classification. We describe the mathematical model for this method as well as present the application for a domain of street-side image datasets. In this application, the detection of façade elements has improved by up to 20% using Multi-view Random Fields.

Keywords

Random Fields, Context, Redundancy, Image Stacks

1. INTRODUCTION

In a current computer vision research input data is often represented as large, redundant datasets with hundreds or even thousands of overlapping images. As the volume and complexity of data increases, it is no longer meaningful to employ manual inputs in any step of the process. This constraint on the work automation leads to a need to utilize as much information from images as possible. One potential approach is to employ "context". Most popular methods of context application are graphical models, specifically Random Fields. However, general Random Fields models are defined such that they allow observations only from a single image. This approach is limiting context as a feature of a single image, but the context is derived from objects in a real scene, from which an image is only one projection. How is this limiting context application and how can we expand the Random Fields model to cope with the presence of multi-view dataset is the topic of this paper.

The basic element in a Random Field model is a "site". This is generally a patch of image area ranging from a single pixel to a larger segment. In our application in a street-side images domain, a site is a rectangular area (block) of a building façade (see Figure 1). Each site has to be labeled according to visual data and a context in which it is observed. Context is defined as relations (spatial relations, similarity…) between sites. In a multi-view scenario, we have multiple matched images, each with its own set of sites. Extension of Random Fields into a multi-view is not straightforward, as the two sets of sites from matched images are typically overlapping.

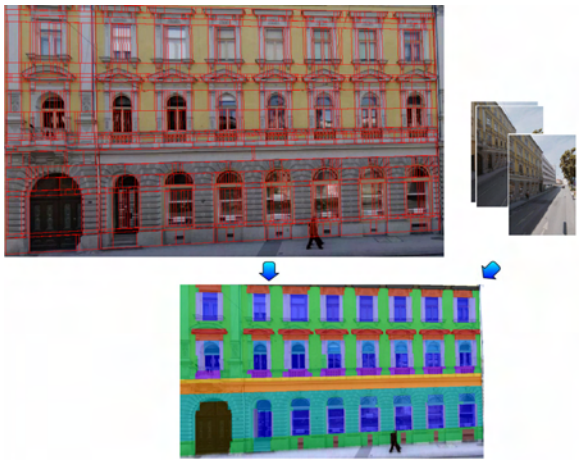


Figure 1: The application of Multi-View Random Fields for labeling of the façade elements. Top left – set of blocks that divide building façade into a set of sites for a graphical model. Bottom – final labeling is achieved as a combination of information from multiple overlapping images (for color-coding, see Figure 7).

Simple merging of these two sets would cause double detections of same objects and unresolved relations between sites. To solve both problems, we introduce a new concept – Multi-View Random Fields.

In this paper, the "*Background"* and "*Graphical Models"* sections are outlining a context of our work in a computer vision community and in a Random Fields models research. The "*Context in Multi-View"* section explains what type of context is relevant in multi-view and how it can be utilized. In the "*Multi-View Random Fields"* section the new graphical model is introduced and the *"Application of MVRF"* section present the illustrational application of the model in a street-side images domain.

2. BACKGROUND

The last decade saw growing interest in multi-view methods. With the introduction of a new generation of high resolution digital cameras and with rapid improvements in storage and computing hardware, multi-view imagery advanced from a source for the computation of point clouds by two-image stereo methods to a broad range of vision problems employing thousands of overlapping images. Open online image hosting sites (Flickr, Picasa, Photobucket…) have added interesting vision opportunities. While the basic principles for matching images remain applicable to such datasets [Har04a] [Leo00a], new problems needed to get solved, such as the organization and alignment of images without any knowledge about camera poses [Sna06a]. The resulting resource need in computing gets addressed by means of graphical processing units GPUs, or with distributed approaches [Fra10a]. Therefore current computer vision can cope with this avalanche of imagery and multi-views are becoming a common reality.

Extending the concept of Random Fields into such multi-view scenario comes from an idea that given more images of the same scene, more contextual relations can be examined. In this work, we present a mathematical model for Multi-View Random Fields that allows transferring contextual relations between matched images. We also present the application of Multi-View Random Fields in a domain of street-side images. This domain is useful for a demonstration, as there are large datasets of matched street-side images for the purpose of urban modeling (virtual cities, GIS, cultural heritage reconstruction) that establish a multi-view scenario. Urban scenes also exhibit strong contextual relations, as man-made objects adhere to an inherent organization. We show how façade elements can be classified, using both context and multi-view principles in one model.

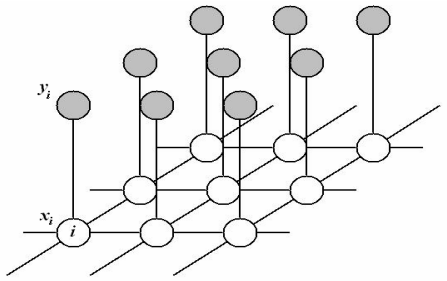


Figure 2. The typical application of MRF in computer vision. At each node (site) *i*, the observed data is denoted as y_i and the corresponding label as x_i. For each node, only local observations are possible. Generally each node represents a pixel in an image and observed data pixel's features.

3. GRAPHICAL MODELS

The most common non-causal graphical models in computer vision are Markov Random Fields (MRF).

MRF have been used extensively in labeling problems for classification tasks in computer vision [Vac11a] and for image synthesis problems. In a labeling task, MRF are considered to be probabilistic functions of observed data in measured sites of the image and labels assigned to each site. Given the observed data $\mathbf{y} = \{\mathbf{y}_i\}_{i \in S}$ from the image, and corresponding labels $\mathbf{x} = \{x_i\}_{i \in S}$, where S is the set of sites, the posterior distribution over labels for MRF can be written as:

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z_m} \exp\left(\sum_{i \in S} \log p(\mathbf{y}_i \mid x_i) + \sum_{i \in S} \sum_{j \in N_i} \beta_m x_i x_j \right), \quad (1)$$

where Z_m is the normalizing constant, β_m is the interaction parameter of the MRF and N_i is the set of neighbors of a site i. The pairwise term $\beta_m x_i x_j$ in MRF can be seen as a smoothing factor. Notice that the pairwise term in MRF uses only labels as variables, but not the observed data from an image. In this arrangement, the context in a form of MRF is limited to be a function of labels, thus allowing for semantic context (context between classes) and limiting geometric context to a structure of MRF graph (see Figure 2). This makes the MRF applicable mainly for simpler forms of local context.

To cope with such limitations, the concept of Conditional Random Fields (CRF) was proposed by J. Lafferty [Laf01a] for the segmentation and labeling of text sequences. The CRF are discriminative models that represent the conditional distribution over labels. Using the Hammersley-Clifford theorem [Ham71a], assuming only pairwise cliques potentials to be nonzero, the conditional distribution in CRF over all labels **x** given the observation **y** can be written as

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z} \exp\left(\sum_{i \in S} A_i(x_i, \mathbf{y}) + \sum_{i \in s} \sum_{j \in N_i} I_{ij}(x_i, x_j, \mathbf{y}) \right), \quad (2)$$

where Z is the normalizing constant, $-A_i$ is the unary and $-I_{ij}$ pairwise potential. The two principal differences between conditional model (2) and MRF distribution (1) are that the unary potential $A_i(x_i, \mathbf{y})$ is a function of all observations instead of only one observation $\mathbf{y}_i$ in a specific site i and the pairwise potential in (2) is also the function of observation, not only labels as in MRF. In CRF, the unary potential $A_i(x_i, \mathbf{y})$ is considered to be a measure of how likely a site i will take label x_i given the observation in a image $\mathbf{y}$. The pairwise term is considered to be a measure of how the labels at neighboring sites i and j should interact given the observed image $\mathbf{y}$. This concept of CRF allows for use of more complex context derived from larger sets of observations in the image and employing geometric context (e.g. spatial relations between objects). It is extended even more in a concept of Discriminative Random Fields [Kum06a], where an arbitrary discriminative classifier can be applied in a form of unary/pairwise potential.

However, in all concepts of Random Fields, the set of sites S (and thus the observations) is limited to a single image. How to extend these models into a multi-view is explained in subsequent sections.

4. CONTEXT IN MULTI-VIEW

Before the definition of a new Random Field model in multi-view, we must consider what type of context can be transferred between images. The most common type of context applied for classification is a local pixel context. In general, a small neighborhood around an examined pixel is taken as a context area and a graph structure of a model is placed in this neighborhood (one node per pixel). However, this approach is not suitable for multi-views, as neighborhoods around matched pixels in two images are in general uniform and will not present much useful additional information. Alternatively we can consider global context, which examines relationships between all objects in the images. In this type of context, we can observe different relations in different images, thus transferring such context would provide additional information for recognition and classification (see Figure 3). If spatial relations between objects are examined in this manner, graphical models are approximating spatial relations between objects in a real 3D scene.

In a standard Random Fields (RF) model, each image is considered a unique unit of information. Thus, we can consider a global context to be a specific feature of each image - the global context is a set of relations between all sites detected in a single image.

Figure 3. Building façade projected in slightly different views. Red lines (graph edges) represent spatial relationships between objects detected in the images, indicating different context in two projections for the same objects. For better overview, only some relations are visualized.

Typically, sites are either *pixels* or *segments*. Construction of a global model with node in each pixel would significantly increase the complexity of computation; therefore we consider segments as the representation of sites in our model.

Subsequently a site is represented by a specific area (segment) in a digital image. Such area represents an object (or part of object) and areas from two sites are not overlapping. In a general RF model, a set of all sites in one graph is denoted as S. In a local model, one set S include sites from a small patch of the image, however in a global model, S includes all sites from the entire image. Visual features of the area assigned to a specific site are denoted as image observation $\boldsymbol{y}_s$ from site $s \in S$. In a graphical model, if there is an edge between nodes assigned to sites s_1 and s_2, let's denote this relation as $\Phi(s_1, s_2) = 1$ and consequently if there is no edge between s_1 and s_2, denote this as $\Phi(s_1, s_2) = 0$.

Transferable Sites

Consider one image from the dataset as "examined image" to which we would like to transfer context from other matched images. Let's call any site $s \in S$ from an examined image a "native site". If the image matching is established in a dataset (we have a set of corresponding points that link images), we can look for any sites from other images that are *corresponding* to native sites. In most cases, sparse point cloud of matched points is enough to establish correspondence between site. Relative poses between images and camera parameters are not required. Definition of corresponding sites can vary in different applications. In general, *corresponding sites are two sites from different images that share some subset of corresponding points*;

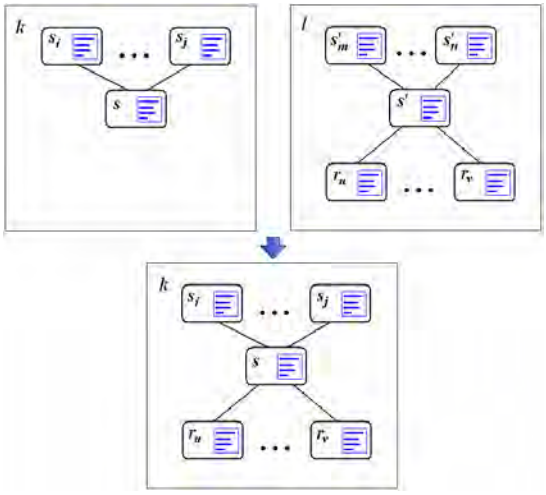


Figure 4. Transfer of sites from the image $l \in$ I to the image $k \in$I, as presented in Definition 1. Only sites from l that are not corresponding to any sites from k are transferred. This figure demonstrates only transfer between two images.

each site from matched images can have only one corresponding site in the examined image – the example of this relation is provided in the application section of this paper.

Given that corresponding sites usually represent the same objects, transferring such information between images would be redundant. Therefore we transfer sites that have no correspondences in the examined images to provide new information. We denote such sites as "transferable sites". For a single, examined image from the image stack, let's define the set of transferable sites as:

Definition 1: *If* $S_k = \{s_1, s_2, \ldots, s_n\}$ *is the set of sites for single image* $k \in$ I, *where* I *is the set of images and correspondences have been established between the images from* I *such that* $s_i' \in S_l$ *is a site from image* $l \in I-\{k\}$ *corresponding to a site* s_i. *Than the* $R_k = \{r_1, r_2, \ldots, r_m\}$ *is the set of transferable sites for the image* k *if* $\forall r_j \in R_k \exists s_i \in S_k \mid \Phi(r_j, s_i') = 1$ *and* $\forall r_j \in R_k \neg \exists r_j' \in S_k$. R_k *is constructed such that* $\forall r_i, r_j \in R_k$, r_i *and* r_j *are not correspondent to each other in any two images from I*

Thus the R_k is the set of sites from other images than k, that are in the relationship in graphical model with some corresponding site to sites from S_k, but themselves have no correspondences in S_k (see Figure 4). The set of transferable sites can be seen as a context information, that is available in the image stack, but not in the examined image. If sites are the representations of objects, than in a transferable set, there are objects in context with the scene of the image that are currently not located in the projection, thus are occluded, out of the view or in different timeframe. This also means that the visual information from the sites in R_k are not present in the image k. If the sites from R_k are included in the vision process, they can provide additional context and visual information that is not originally present in the examined image.

Note that a transferable site is not equivalent to a native site in an examined image. Even though transferable sites have the same set of visual features as sites native to the image and they can be assigned the same set of spatial and contextual relations in a graphical model, transferable sites lost all original contextual relationships except the relationships to the sites they are connected within the examined image. This makes them harder to label. But the labeling of transferable sites is not the aim in the case of examined image (the goal is to label only native sites), thus transferable sites can contribute information for image labeling, but the labeling of themselves is usually irrelevant.

5. MULTI-VIEW RANDOM FIELDS

Given a non-equality of transferable sites to native sites, standard RF models are not compatible with this extended set. For this reason, we introduce a new model denoted as *Multi-View Random Fields* (MVRF). This model is derived from a CRF, described in Section 2; however we extend the posterior probability distribution into MVRF model framework as follows:

Given the observed data $\mathbf{y} = \{\mathbf{y}_i\}_{i \in S}$ from the image, corresponding labels $\mathbf{x} = \{x_i\}_{i \in S}$, where S is the set of native sites from the image and observations from transferable set $\mathbf{z} = \{\mathbf{z}_i\}_{i \in R}$ with corresponding labels $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i \in R}$, where R is the set of transferable sites, the posterior distribution over labels is defined as:

$$P(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = \frac{1}{Z} \exp\left(\sum_{i \in S} A_i(x_i, \mathbf{y}) + \sum_{i \in R} A_i'(\tilde{x}_i, \mathbf{z}_i) + \right.$$
$$\left. + \sum_{i \in S} \left(\sum_{j \in N_i} I_{ij}(x_i, x_j, \mathbf{y}) + \sum_{j \in K_i} I_{ij}'(x_i, \tilde{x}_j, \mathbf{y}, \mathbf{z}_j) \right) \right) \quad ,(3)$$

where Z is the normalizing constant, N_i is the set of native sites neighboring site i and K_i is the set of transferable sites neighboring site i. - A_i and - A_i' are unary potentials, - I_{ij} and - I_{ij}' are pairwise potentials (for native sites and transferable sites respectively). The differences between potentials for transferable sites and for native sites are as follows:

- In the unary potential for a transferable site, only observations from the site itself are

considered, instead of observation from the entire image for native sites. This is due to the fact, that a transferable site does not have any connections to the image except for the site it is neighboring. Even if other connections exist (with other sites in the image), it is a hard task to establish relationships. For native site, there are no changes to a standard conditional model.

- In the pairwise potential, in addition to observation from the image, local observation from the transferable site is considered, when relations are examined between a native site and transferable site. The inclusion of all image observation grant at least the same level of information in pairwise computation as in a standard CRF model and the additional observation form transferable site represent extended context for native image observation. The pairwise potential for two native sites is the same as in a standard CRF model.

This model has some additional unique characteristics. For example, no pairwise relations are considered between two transferable sites. This is based on the construction of transferable sites set. A site from such set can be neighboring several native sites, but not any other transferable site. This can be seen as a limitation for the model, however without additional high frequency information about the scene (as a prior knowledge), it is virtually impossible to establish relationships for transferable sites.

The computational complexity of the model is not increased significantly. Pairwise potentials are computed only for native sites, as it is in the standard CRF model. The difference is in the number of neighbors for each site, however even this number should not increase significantly. When considering a global model, each new neighbor (transferable site in relation to the native site) represents a new object in the projection. This is dependent on the differences between projection parameters – camera positions, optical axes…, but even for very different parameters, the number of objects should not differ significantly for the same scene. From the general observation, the number of neighboring transferable sites is notably lower than the number of neighboring native sites.

Potentials Modifications

Unary potential for native image sites, similar to a standard CRF is a measure of how likely a site i will take label x_i given the observations in image $\mathbf{y}$. A standard approach described in a work of S. Kumar is to apply Generalized Linear Models (GLM) as local class conditional [Kum06]. In that case, given the model parameter $\mathbf{w}$ and a transformed feature vector at each site $\boldsymbol{h}_i(\mathbf{y})$, the unary potential can be written as:

$$A_i(x_i, \mathbf{y}) = \boldsymbol{log}\left(\sigma\left(x_i \mathbf{w}^T \boldsymbol{h}_i(\mathbf{y})\right)\right) \quad ,(4)$$

For the transferable sites, the feature vector is limited to the observations from single site. This limitation defines a new expression for unary potential, exclusive to transferable sites as

$$A_i'(\tilde{x}_i, \mathbf{z}_i) = \boldsymbol{log}\left(\sigma\left(\tilde{x}_i \mathbf{w}^T \boldsymbol{h}_i(\mathbf{z}_i)\right)\right) \quad ,(5)$$

The feature vector $\boldsymbol{h}_i(\mathbf{z}_i)$ at the transferable site i is defined as a nonlinear mapping of site feature vectors into high dimensional space. The model parameter $\mathbf{w} = \{w_0, \mathbf{w}_1\}$ is composed of bias parameter w_0 and model vector $\mathbf{w}_1$. $\sigma(.)$ is a local class conditional, that can be any probabilistic discriminative classifier.

The pairwise potential for two native sites from the image remains the same as in CRF model, given the GLM are applied to compute the class conditional:

$$I_{ij}(x_i, x_j, \mathbf{y}) = \beta\left(K x_i x_j + (1-K)\left(2\sigma\left(x_i x_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y})\right) - 1\right)\right) \quad ,(6)$$

where $0 \le K \le 1$, $\mathbf{v}$ and β are the model parameters and $\boldsymbol{\mu}_{ij}(\mathbf{y})$ is a feature vector. For transferable sites, we introduce the additional feature vector in a form of observations from specific site:

$$I_{ij}'(x_i, \tilde{x}_j, \mathbf{y}, \mathbf{z}_j) = \beta\left(K x_i \tilde{x}_j + (1-K)\left(2\sigma\left(x_i \tilde{x}_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y}, \mathbf{z}_j)\right) - 1\right)\right) \quad ,(7)$$

where $\mu_{ij}(\mathbf{y},\mathbf{z}_i)$ is a feature vector defined in a domain $\mu : \Re^{\gamma} \times \Re^{\gamma} \to \Re^{q}$ such that observations are mapped from the image/sites related to site s into a feature vector with dimension γ. Note that the smoothing term $K x_i \tilde{x}_j$ is the same as in a standard CRF definition. Thus if $K = 1$, the pairwise potential still performs the same function as in a MRF model, however given new transferable sites, the smoothing function will depend also on their classification $\tilde{x}_j$. In this case, visual information from transferable sites is not involved in the pairwise term and is only applied in the unary term. If $K<1$ the data-dependent term $2\sigma\left(x_i \tilde{x}_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y}, \mathbf{z}_j)\right) - 1$ is included in a pairwise potential. Observations from the image related to the examined native site and observation from transferable site are transformed into feature vector and involved in computation.

Parameter Learning and Inference

In this work, we constructed an MVRF model to be as compatible with other RF models as possible. This approach is observed also in a parameter learning process, as any standard method used for learning of

CRF model can be also used for MRVF model. To further simplify the process, we observed that learning from single (un-matched) images is feasible without the loss of strength of the model. This is due to the construction of potentials - in a unary potential, visual features do not change for transferable sites, therefore they can be learned directly from single images in training dataset. The spatial relations defined for a pairwise potential also do not change significantly for the pair native-transferable site. For such reasons, we can assume that the MVRF model can be learned even directly from single images without dataset matching. Therefore, methods such as pseudo-likelihood can be applied for learning.

Similarly, parameter inference can be performed, using any standard method applied in CRF. In our application, we use Belief Propagation, but other possible methods are Tree-Based Reparameterization or Expectation Propagation for example.

6. APPLICATION OF MVRF

In this section we present the application of MVRF in the building façades dataset for the purpose of façade elements detection and classification. This application is based on the dataset provided by a vehicle-based urban mapping platform. Sparse image matching is applied (see Figure 5), using the Structure-from-Motion method [Irs07a]. We selected the left camera subset, since it provides a clear view of the building façades, not distorted by the perspective (which, however, is easy to rectify) and with good visual cues. This setting will demonstrate the advantages of MVRF in cases when a site was misdetected and presents lost contextual information in standard models. In most images, the building façade is not projected in its entirety and parts are located in other images. Therefore in such cases, the MVRF will also provide new contextual and visual information in a form of transferable sites based on the objects that are not located in the original image.

In each image, separate facades are detected. This can be achieved when the wire-frame models of the scene are available, or using visual cues, such as repetitive patterns [Rec11a]. Subsequently, a modified gradient projection is applied to segment each façade into a set of blocks. This method is based on a standard gradient projection approach [Lee04a] designed for the detection of windows with following modifications:

First, we vertically project gradients to establish a horizontal division of the façade into levels (level borders are located at the spikes of the projection). Subsequently, we compute horizontal gradient projections in each level separately.

Figure 5. Top row: two examples of the same façade, matched with a sparse point cloud (red dots). Middle row: set of blocks located in each façade (left image show façade detail for better overview, right image entire facade). Bottom row: set of blocks from the first image projected into a second image and a set of transferable sites (highlighted blocks) that is derived from the projection (as sites that have no correspondence in second set).

This process will yield a set of blocks bordered by level borders horizontally and spikes in projection vertically (see Figure 5). Second, we consider each block as a site for a graphical model, thus we compute visual features for each block and consider spatial relationships between blocks. Visual features, such as texture covariance, or clustering in a color space are used for classification [Rec10a]. For example, clusters in a CIE-Lab color space are computed for each block and are compared to class descriptors.

When the segmentation of a façade into a set of block is established, we can define a global graphical model in this structure. Each block is considered a site, thus each node of the graph is placed in a separate block. We define neighborhood relation such that for each block, its neighbors are all blocks located in areas above, below, left and right from itself (see Figure 6). This definition allows considering all objects at the same level and column to be involved in contextual relations, accounting for relations, such as rows and columns of windows, or window-arch. An edge of a graphical model is placed between each two neighboring blocks. In this approach, a separate graph is created for each façade in the image.

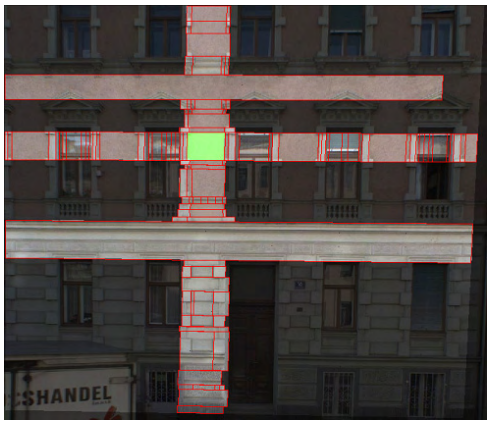


Figure 6. Example of site neighborhood, as defined in this application. Green block is the examined site and highlighted blocks are defined as its neighborhood.

Multi-View Scenario

To establish a multi-view, we use a sparse point cloud. We match blocks between images such that we interpolate between detected corresponding points to achieve rough point-to-point matching. If two blocks in different images share at least 2/3 of matched points (detected and interpolated), we define these as corresponding blocks. Given one image as "examined", we can label all blocks from the same façade in other images as either corresponding or non-corresponding. Subsequently, transferable sites are blocks that are from the same façade as in an examined image, but are non-corresponding to any block from the examined set (see Figure 5). Establishing the relations between native and transferable sites is straightforward, as we can still consider up, down, left, right directions. With these definitions, we can construct the MVRF model from our dataset.

Experiments

We use the described model for the purpose of façade elements detection and classification. The set of classes with corresponding color coding is displayed in Figure 7. Our testing dataset consists of 44 matched images. This dataset covers three full building façades and one half façade. A sparse point cloud of 1429 3D points is used to match images. Approximately 800 points are projected into each image. In the testing process, we compare the number of façade elements to the number of detected elements with the applied method. We counted overall numbers of elements through the entire dataset, as displayed in Table 1. For example, total number of 536 "window centre" elements can be observed in all images, that is approximately 12 "window centers" per image.

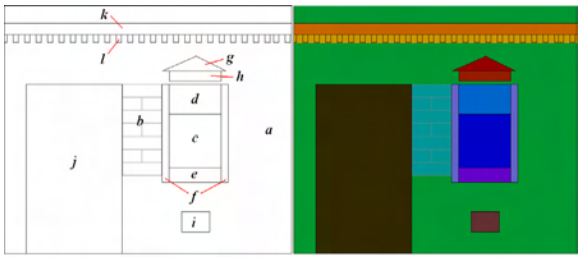


Figure 7. Set of classes: *a*) clear façade; *b*) brick façade; *c*) window centre; *d*) window top; *e*) window bottom; *f*) window margins; *g*) arch top; *h*) arch bottom; *i*) basement window; *j*) door; *k*) ledge; *l*) ledge ornament; On the right side, color representation of each class is displayed.

Each façade was processed separately, that is if there were two façades in one image, such image was processed two times (each time for different façade). After running the algorithm, a number of detected elements is counted visually. The façade element is defined as *detected*, if at least 2/3 of its area is labeled with the corresponding class. For the training purpose, we used the subset of 3 images from the dataset and other 5 unrelated images as labeled ground truth. This proved to be sufficient, as the spatial relations between classes are in general stable through different facades and a certain visual features variability

Class	# el	single	multi /native	multi /trans
clear façade	61	61	61	61
brick façade	54	54	54	54
win. centre	536	485	531	531
window top	311	270	303	308
win. bottom	300	227	273	288
win. margin	683	572	618	654
Arch top	199	176	189	192
Arch bottom	199	184	194	194
Basem. win	121	98	115	117
Door	34	32	33	33
Ledge	90	90	90	90
Ledge orna.	34	32	34	34

Table 1. The Results for the MVRF application. "# el" displays the overall number of each class for entire dataset (44 images). "single" displays detected elements in MVRF single image scenario (equivalent to CRF), "multi/native" displays results for multi-view scenario with only native sites in results and "multi/trans" display results for multi-view scenario with transferable sites labels in results. Numbers displayed are the detected façade elements in all images of dataset.

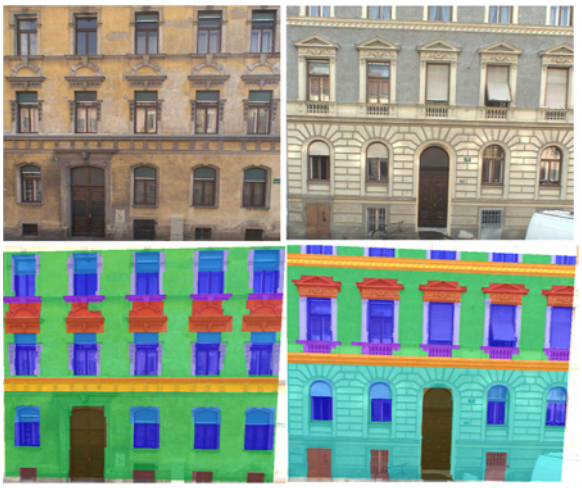


Figure 8. Two examples of classification results. Classes are labeled according to color scheme explained in Figure 7. Colors are superimposed over original images in the bottom row.

was allowed by the use of descriptors (e.g. clustering). We trained on single images without the use of matching. For the parameter inference, we used a Belief Propagation method. Initial classification was performed based on only visual features and in each iterative step of the method, it was refined by pairwise relations and site features described in a model. In each step, we also refined visual descriptors for each class to better approximate features in each unique façade. Results can be observed in the Table 1. We included results for scenarios, where no transferable sites were used (single), and the MVRF model is equivalent to CRF in this case, results when only labels of native sites were considered and results were labels of transferable sites were included. Notice a significant improvement in detection for classes that are visually ambiguous, but have strong contextual relations (e.g. window margins, window tops). For a "win. bottom" class, the correct detection rate improved from 76% in a single-view to a 96% in a multi-view with transferable sites projected, thus achieving a 20% improvement. Results illustrated in Figure 8.

7. CONCLUSION

In this paper, we addressed a common problem in a current research – how to work with context information in matched datasets and to alleviate an artificial limitation of graphical models to single images. We introduced a new MVRF model directly applicable in a multi-view scenario. We extended the standard CRF model such that it can work with overall context of the scene present in the multi-view dataset, but it still retains the same properties for processing visual and contextual information in a single image. Validity of this model is subsequently demonstrated in the application in street-side image domain – detection of façade elements. However the new MVRF model is applicable in same situations as a standard CRF model, provided that appropriate image matching is available. For example, the MVRF model was also used for a super-pixel based semantic segmentation of outdoor images in our other work.

8. REFERENCES

[Fra10a] Frahm, J. M., et al. Building Rome on a Cloudless Day. *European Conference on Computer Vision,* pp. 368–381, 2010

[Ham71a] Hammersley, J. M., Clifford, P. *Markov* field on finite graph and lattices. Unpublished, 1971

[Har04a] Hartley, R. and Zisserman, A. Multiple View Geometry in Computer Vision. *Cambridge University Press*, ISBN: 0521540518, 2004

[Irs07a] Irschara, A., et al. Towards wiki-based dense city modeling. *International Conference on Computer Vision*, pp. 1-8, 2007

[Kum06a] Kumar, S. and Herbert, M. Discriminative random fields. *International Journal of Computer Vision*, 68(2), pp. 179–201, 2006

[Laf01a] Lafferty, J., et al. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*, pp. 282-289, 2001

[Lee04a] Lee, S. C. and Nevatia, R. Extraction and integration of window in a 3d building model from ground view images. *Computer Vision and Pattern Recognition*, pp. 112-120, 2004

[Leo00a] Leonardis, A., et al. Confluence of computer vision and computer graphics. NATO science series, *Kluwer Academic Publishers*, ISBN 0-7923-6612-3, 2000

[Rec11a] Recky, M,. et al. Façade Segmentation in a Multi-View Scenario. *International Symposium on 3D Data Processing, Visualization and Transmission*, pp. 358-365, 2011

[Rec10a] Recky, M. and Leberl, F. Windows Detection Using K-means in CIE-Lab Color Space. *Internation Conference on Pattern Recognition,* pp. 356-360, 2010

[Sna06a] Snavely, N., et al. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics*, pp. 835 – 846, 2006

[Vac11a] Vacha, P., et al. Colour and rotation invariant textural features based on Markov random fields. *Physical Review Letters, No. 6*, pp. 771-779, 2011

Collision Detection on Point Clouds Using a 2.5+D Image-Based Approach

Rafael Kuffner dos Anjos
Inesc-ID
Av. Prof. Dr Anibal Cavaco Silva
Portugal 2744-016, Porto Salvo
rafaelkuffner@gmail.com

João Madeiras Pereira
IST/Inesc-ID
Rua Alves Redol, 9
Portugal 1000-029, Lisboa,
jap@inesc-id.pt

João Fradinho Oliveira
C3i/Inst. Politécnico de Portalegre
Praça do Município
Portugal 7300, Portalegre
joaofradinhooliveira@gmail.com

ABSTRACT

This work explores an alternative approach to the problem of collision detection using images instead of geometry to represent complex polygonal environments and buildings derived from laser scan data, used in an interactive navigation scenario. In a preprocessing step, models that are not point clouds, are sampled to create representative point clouds. Our algorithm then creates several 2.5+D maps in a given volume that stacked together form a 3D section of the world. We show that our new representation allows for realistic and fast collision queries with complex geometry such as stairs and that the algorithm is insensitive to the size of the input point cloud at run-time.

Keywords

Collision Detection, Point-based Graphics, Navigation

1. INTRODUCTION

Collision detection is normally a bottleneck in the visualization and interaction process, as collisions need to be checked at each frame. Traditionally, the more complicated and crowded is our scene, the more calculations need to be done, bringing the frame-rate down. Therefore the optimization of this process, gaining speed without losing quality in the simulation, is something that has been researched for years. Although several different techniques and approaches have been developed, and showed good performance in specific scenarios, these approaches rely on object topology information which is easily extracted from polygonal models. However with point cloud models, the classical approaches either will not work, or will have to heavily adapt to this specific scenario, compromising their optimizations.

Using images as an alternative way of executing the task of collision detection might just be the answer. Image-based techniques can have their precision easily controlled by the resolution of the used images, and the algorithms are completely independent of the object's topology and complexity at run-time. It does not matter whether an object has sharp features, round parts, or even whether it is a point cloud, as all we are dealing with is the object's image representation. Being a scalable and promising technique, Image-based collision detection seems to be a plausible alternative to the classical approaches.

Our approach focuses in a virtual reality navigation scenario, where the scene is derived from the real world via devices such as laser scanners, which tend to generate enormous point clouds. Also, the hardware at hand might not fit the basic requirements for most algorithms and techniques, a situation that commonly will happen in tourism hotspots, museums, or other places where we would like ordinary people to be able to interact with the system. The developed application enables them to control an avatar on a static environment, a point cloud or polygonal model.

The main contribution of our research is a new 3D world representation for environment and buildings which is completely image-based with information that enables realistic and robust collision detection with underlying complex geometry such as slopes and stairs. Slicing a given structure along the Z axis (Using Cartesian coordinates as illustrated in Figure 1); we create a discrete set of images containing height information about the surface, and possible collidable frontiers. It is a flexible format that is able to represent either point clouds or polygonal models. This representation allows us to perform collision detection with user chosen precision, and high scalability.

2. RELATED WORK

The problem of collision detection is present in several areas of research and applications, each of them having different concerns, requirements and desired results. This necessity has prompted the creation of several techniques that try to deliver these results using varied approaches, each one fitting to specific problems.

Feature based: Famous examples are the Lin-Canny algorithm [MLJC] and its more recent related algorithm, V-Clip [BMM], which keeps track of the closest features between two objects, deriving both the separation distance, and the vertices that have possibly already penetrated the other object.

Bounding Volume Hierarchies: Different volumes have been developed to envelop the basic features, such as Spheres [PMH], Oriented Bounding Boxes (OBB) [SGMCLDM], or Axis Aligned Bounding boxes (AABB) [TLTAM] [MJBJ] [XZYJK], each of them has its advantages over the others; Spheres are easier to fit, OBBs have faster pruning capabilities, and AABBs are quicker to update, therefore being very popular in deformable body simulation.

Also, different tree traversing and creation techniques [TLTAM] [SGMCLDM] have been developed to optimize these expensive operations, taking into account each specific kind of application.

Stochastic algorithms: Techniques that try to give a faster but less exact answer have been developed, giving the developer the option to trade accuracy in collisions with computing power. The technique based on randomly selected primitives, selects random pairs of features that are probable to collide, and calculates the distance between them. The local minima is kept for the next step and calculations are once again made. The exact collision pairs are derived with Lin-Canny [MLJC] feature based algorithm. With a similar idea, Kimmerle et. al [SKMNFF] have applied BVH's with lazy hierarchy updates and stochastic techniques to deformable objects and cloth, where not every bounding box is verified for collision, but it has a certain probability.

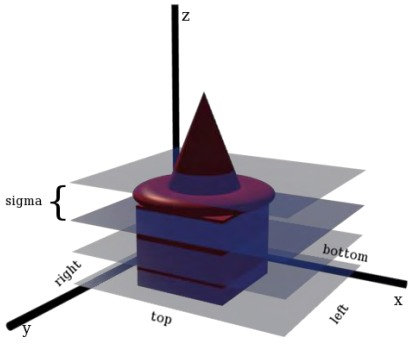


Figure 1. Slices creation process, and camera variables.

Image-based algorithms: These solutions commonly work with the projection of the objects, in contrast with previous techniques that work in object space, meaning that they are not dependent of the input structure, and as such are more suitable to point clouds. However to our knowledge they have not yet been applied to point clouds.

RECODE [GBWHS] and several other works [DKDP] [KMOOTK], [GBWSW] take advantage of the stencil buffer and perform collision detection on it by using object coordinates as masks, and thus detecting possible penetrations.

CULLIDE [NSRMLDM] uses occlusion queries only to detect potentially colliding objects, and then triangle intersection is made on the CPU. They render the bounding volumes of the objects in normal and reverse list storage order, and remove the objects that are fully visible in both passes, meaning that these objects are not involved in any collision.

Heidelberger et. al [BHMTMG] uses simple AABB's as bounding volumes for the objects in the scene. Potentially colliding objects are detected, and a LDI (Layered Depth Image [JSLR]) of the intersection volume between the two objects is created. That is, a volumetric representation of an object across a chosen axis. At each rendering step, as a polygon is projected into the LDI, the size of the intersubsection volume is computed. Faure et. al [FSJF] [JFHFCP] addresses not only collision detection, but also its response by using this same principle.

On a collision avoidance scenario, we want to predict an upcoming collision, and use this information to prevent it from happening. It is used mostly in artificial intelligence to control intelligent agents or robots. Loscos et. al [CLFTYC] represent the environment as patches to where the agent can or cannot go according to its occupation. It can be considered a basic but robust image based approach since it uses a 2D map to represent the environment.

Collision detection on point clouds: Algorithms using feature based techniques, bounding volumes, and spatial subdivision have been developed. Klein and Zachmann [JKGZ] create bounding volumes on groups of points so collision detection can be normally applied. Figueiredo et. al [MJBJ] uses spatial subdivision to group points in the same voxel, and BVHs to perform collision detection. The main issue while dealing with point clouds is the absence of closed surfaces and object boundaries. Ordinary BVH or stochastic techniques have to heavily adapt to this scenario, normally leading to not so efficient hierarchies. Feature-based techniques that work at vertex level are not scalable enough to be suited to these scenarios, since point clouds are normally massive data sets. Image-based techniques have the

```
for all points p in m do
    s ← floor((abs(z − z_min)) / σ)
    red ← (abs(z − z_min) mod σ) / σ
    red_old ← cube[x_screen][y_screen][s]
    if (abs(red_old − red)) / σ > σϵ then
        cube[x_screen][y_screen][s] ← 1
        p.color(red, 1, 1)
        if red_old > red then
            p.z ← p.z + (red_old − red) * σ + ϵ_2
        end if
    else
        cube[x_screen][y_screen][s] ← red
        p.color(red, 1, 0.1)
    end if
end for
```

Figure 2. Points Coloring and Obstacle Detection Algorithm

disadvantage of sometimes requiring special graphic card capabilities, but only for some implementations. Overall, their properties make them the best suited for the proposed scenario of the tourism kiosk.

For further information on collision detection and avoidance techniques we suggest the following surveys: [NMAS] [SKTGKICB] [MT].

3. IMPLEMENTATION

Image-based algorithms that have been presented in the community ([NSRMLDM] [GBWHS] [DKDP] [NBJM] [JFHFCP] [FSJF]) perform very well in various kinds of scenarios, but some features of our set scenario (described on Section 1) make them hard or impossible to be applied (e.g. our data is unstructured, not all objects are closed or convex).

Our work extends the idea of a 2.5D map presented on the work of Loscos et. al [CLFTYC] by using enhanced height-maps, where the pixel color not only represents the height on that point, but also contains obstacle information, while at the same time overcoming the limitations of only supporting single floor environments. We call these enhanced maps, 2.5+D maps. Instead of having just one height map, we create a series of enhanced maps along intervals sized σ on the z axis, thus enabling the storage of more than a single z value for each (x, y) pair. Unlike LDI's, our representation does not combine several images into a volumetric representation, but separates each slice into a single image so we can easily perform memory management, and apply image comparison and compression techniques to have a better performance. Using the color of each pixel as a representation of a voxel, we write height information on the red channel, and identify obstacles on the blue channel. By adding these variables, we can determine not only the height level where the avatar should be standing, but also if he is colliding with any obstacle in several different heights.

Slices Creation

The creation of this representation is executed in a pre-processing stage, which is composed of several steps that must be performed from the input of the model until the actual rendering to create the snapshots that will be the used as collision maps. These slices are created as following. The camera is first set up according to the previously calculated bounding boxes of the input model on an orthogonal projection. After rendering that projection, a snapshot sized σ is created and saved into the disk for further use. The camera then is moved along the z axis, and the process is repeated until the whole extension of the model has been rendered into images. A visual representation of the described variables along with the slices computed on an example model can be seen on Figure 1 and two real slices can be seen on Figure 4.

Polygonal Model Oversampling

We aim for a solution that accepts both polygonal models and point clouds. However these representations are inherently different and cannot be processed initially in the same way. Hence we created a solution that approximates the polygon models with a synthetic point cloud thus enabling later steps to be processed in the same way. We apply a simple oversampling operation that operates at triangle level transforming a polygonal model into a point cloud with a user-choice level of precision. After oversampling and discarding the polygons, the rendering of the produced point cloud has exactly the same shape and fill as if rendering the polygonal representation to the height map.

Information Processing and Encoding

Since all of our collision information will be written on collision maps as colors, we must assign each point on the point cloud a color representing its collision information. This will not replace the original color of that point in question. When writing these points on the output image, each pixel will represent a voxel sized (t, t, σ) on object space. So the painted color on that pixel will represent all the points contained in that volume. The algorithm on Figure 2 performs both the operation of height map information encoding, and obstacle detection. We define σ as $3t$, so as to ensure that one has more than one point on each slice, to properly perform the obstacle detection, as will be described later.

The first operation is executed as follows: We calculate the difference between the current point z coordinate and the model's lower bounding box z_{min}, and apply the modulo operator with σ. This remainder r represents the points z coordinate on an

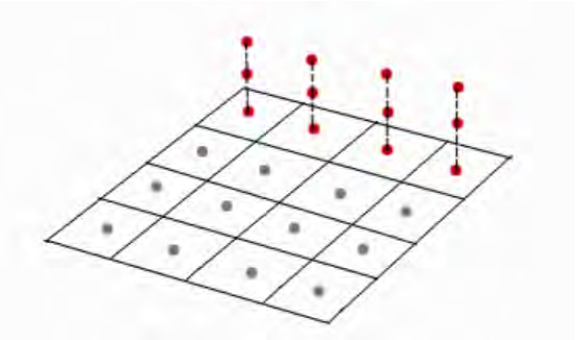


Figure 3. Technique for surface orientation detection. Red points belong to a vertical structure, grey points to the floor.

interval $[0, \sigma]$. To be used as a color value, this difference must belong to the interval [0,1], so we calculate r/σ thus deriving finally the red channel value. The simplified formula is given by $red = \frac{abs(z - z_{min}) mod\, \sigma}{\sigma}$

As navigation on a real-world scenario is normally performed horizontally on the xy plane, we classify an obstacle as a surface that is close to being perpendicular to xy, parallel to zy. So our obstacle collision technique simply estimates how parallel to the z axis a certain surface is. Figure 3 illustrates how potential obstacles and floor are classified using the coloring algorithm (Figure 2). Points lined up vertically on the same pixel most likely belong to a vertical surface. Diagonal structures like ramps are climbable up to a certain degree. The closer to a wall they are, the greater the probability is that its points are considered to be obstacles.

In order to keep track of point information that will be stored in the slices we use an auxiliary structure, a 3D array $cube[w][h][\sigma]$, after processing each point, we keep its color value on the position of the array representing the voxel on object space from where it came from. If there is already a stored value in this voxel, the difference between both red values is calculated, and transformed into an object-space distance $\frac{abs(red_{old} - red)}{\sigma}$.

If this difference is bigger than a certain small percentage ε (e.g. 7%) of the size σ of the slice, we assume that the points are vertically aligned, belonging to a vertical surface. These points are marked on their blue channel with the value 1, and we slightly increase its z coordinate so that the point is not occluded when rendering the maps. Typical output slices produced in the pre-processing stage can be seen in Figure 4, an office environment, where the floor has been correctly labeled as green, and all the walls are labeled as white or light blue.

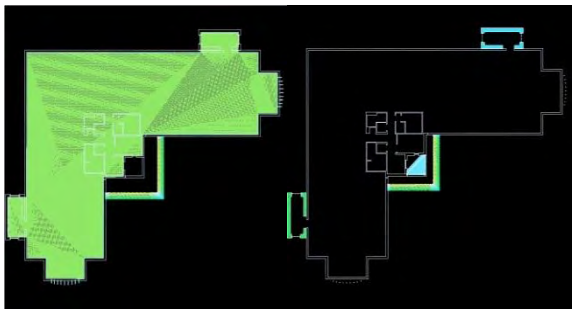


Figure 4. Two slices of an office environment, where walls (white/blue) and floor (green) are clearly distinguished, as well as a subsection of a roof (green to yellow) on the entrance.

Collision Detection

The developed representation provides us with enough information to perform quick collision detection on the navigation scenario given on section 1 where we consider point clouds as a viable input. In the accompanying video we show precise collision detection between rigid bodies and point clouds.

We divide the task of collision detection into two steps: a first step, that we call Broad phase, where we verify the occurrence of collisions between any objects in the scene, and a second step called narrow phase, where we perform collision response.

3.1.1 Broad Phase and Collision Detection

This task consists on identifying possible collisions between all objects on the scene. By representing the avatar that will be navigating on the environment by an Axis Aligned Bounding Box (AABB), we first calculate its size in pixels by calculating $pix_x = \frac{size_x}{t}$ and $pix_y = \frac{size_y}{t}$, where threshold t is calculated as the pixel size. This will be the number of pixels checked for collision on each slice, around the center of the avatar. If any checked pixel is not black, we mark the object as colliding, and they will be further processed in a narrow phase.

The only images we will need to load into the memory at the same time in order to perform collision detection between the given avatar and the environment are the ones located between $slice_0 = \frac{zp_{min} + zp - z_{min}}{\sigma}$ and $slice_n = \frac{zp_{max} + zp - z_{min}}{\sigma}$, where zp represents the z coordinate of the avatar. These slices contain collision detection information about the location where the pawn currently is.

New slices that are needed, are loaded into memory until a user defined constant of number of slices (n_{slices}) is reached. New slices beyond this point, replace an already loaded slice that has the furthest z value from the avatar's own z value, meaning it is not needed at this point of the execution. In practice we found that six slices covered well the avatar's

potential waist, shoulders, head, knees, and feet collisions with the scene.

3.1.2 Narrow Phase and Collision Response

In our current version, the sole purpose of our collision response algorithm was to simply avoid objects from overlapping, and provide a basic navigation experience on the given environment. Any other more complex technique could be applied here, but this simple solution fulfills the requirements for our navigation scenario. We focused on an efficient broad phase algorithm, and a flexible representation so we could apply any chosen image-based technique on the narrow phase. This was achieved with a straightforward extension of our broad-phase algorithm, by applying the concepts of collision response from height maps, and collision avoidance [CLFTYC]. Instead of simply returning true when we find pixels that are not black, we gather information for collision response each time we find colored pixels. Pixels with the blue channel set to 1 always represent an obstacle, except on applications where we want to enable the avatar to climb small obstacles, as the agents from Loscos et.al [CLFTYC]. On these situations, we may ignore these pixels up until the height we want to be considered as climbable. As our avatar moves on fixed length steps, and each time it collides we correct it to the place it was on the previous check, we thus ensure that the avatar is always on a valid position. We apply this (x, y) correction each time an obstacle pixel is found until all the pixels representing the avatar's bounding box are verified.

Height is defined exactly as it is when precomputing height maps. By multiplying the coded height information on the red channel by σ and adding the z base coordinate of the given slice, we have precise information about a given point's height. Collision response can be made by setting the final height to the average height of the points on the base of the bounding box, or by the adopted strategy, the maximum value. Here we also check for surface height values from the first slice until the height we want to consider as climbable.

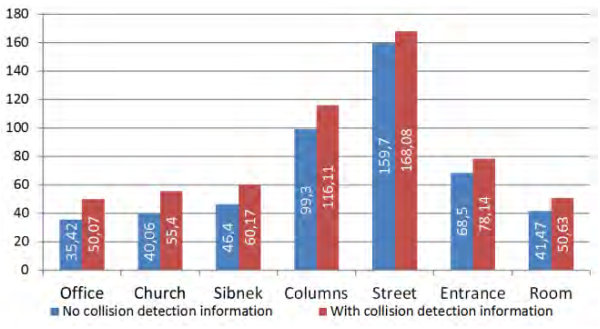


Figure 5. Memory load at a given moment during runtime on a 700x700 configuration, 6 slices, with and without collision detection.

The complexity of this operation is exactly $O(pix_x * pix_y * s)$ where pix_x and pix_y are the number of pixels ocuppied by the base of the avatar and s is the number of slices encompassed by the avatar's height. We point however, that these checks are already performed in the broad phase, and hence can be re-used in the narrow phase without adding any extra operations.

4. EXPERIMENTAL RESULTS

We have implemented the whole algorithm using OpenGL 2.1.2, C and the OpenGL Utility Toolkit

(GLUT) to deal with user input and the base application loop management. The platform used for tests was a computer with an Intel core 2 Duo CPU at 2 GHz with 2GB of RAM, a NVIDIA GeForce 9400 adapter, running Microsoft Windows Seven x86.

Table 2 shows the time taken on the whole preprocessing stage for each model and configuration. Polygon sampling during the preprocessing of polygonal models is clearly the most computationally intensive task in the pipeline, as the two higher complexity point cloud models (Entrance and Room as seen on Table 1 and Figure 7) that did not require sampling had a faster performance. In the preprocessing phase the increase on processing time with point clouds is linear to point complexity. This linear growth is expected since each point must be checked for coloring once, and also for common input processing tasks such as file reading and display list creation.

Regarding the results of the overall memory cost specifically in the preprocessing phase, Table 2 shows that memory scales almost linearly according to the input size of the point cloud (Entrance versus Room in Table 1). This memory is needed temporarily for allocating the auxiliary 3D array for obstacle detection in the slice creation phase. Similarly, tests have shown that this memory consumption also grows linearly with the number of points produced in the polygon sampling phase.

During the application runtime, the average memory consumption varies according to the number of loaded slices into RAM, and according to the size of the loaded model used for rendering (Figure 5 and Table 1). On a 700x700 resolution, the peak minimal value found in any model we experimented was 50,07MB (Office) and the peak maximum 168,08MB (Street), with 6 slices loaded in memory. Table 2 shows how much memory the application consumes when only rendering the models and the total memory with collision detection, while having 6 slices loaded in memory. By controlling n_{slices} we can avoid this value from going over the memory we wish the

Model	Type	Original Complexity	Details
Office	Polygonal	17.353 pts.	Office environment with cubicles and hallways
Church	Polygonal	26.721 pts.	Simple church with stairs and columns
Sibenik	Polygonal	47.658 pts.	Cathedral of St. James on Sibenik, Croatia
Columns	Polygonal	143.591 pts.	Big environment with localized complexity.
Room	3D laser scan	271.731 pts.	3D Scan of a room with chairs and several objects.
Street	Polygonal	281.169 pts.	Outside street environment with an irregular floor, houses, and several objects
Entrance	3D laser scan	580.062 pts.	Entrance of the Batalha monastery in Portugal.

Table 1. Features of the models used for evaluation

application to consume, since all other memory required for the application is for tasks unrelated to collision detection.

We were interested in studying the direct behavior of our algorithm and did not wish to mask performance with asynchronous I/O threads. Results on collision detection have been verified through establishing a fixed route to navigate with the pawn avatar where it goes through different situations and scenarios, such as "climbing" single steps, "traversing" planar areas, going "near" cylindrical or square shaped columns and "falling" from a height. Whilst the number of created collision maps for a scene can affect collision results, the actual number of buffer slices n_{slices} will only affect potentially the performance, as the system checks all slices at the avatar´s current height. Tests on Cathedral 700x700 with 130 slices and n_{slices} set to 10 have showed us that reading from the disk at runtime has a bigger impact on efficiency than storing a higher number of images on commodity RAM. For instance, when using a maximum buffer of 10 slices and reading a high resolution image from the disk, we notice a sudden drop in frame-rate (Figure 6), and this can also be noticed when the pawn falls from a higher structure, and needs to rapidly load a series of maps on his way to the ground. By increasing n_{slices} to 16 on this scenario, we ensure the storage of enough slices to represent the ground floor and the platform on top of the steps (Experiment B in the accompanying video). Little difference was noticed on memory load (5,33MB), and the interaction was much smoother.

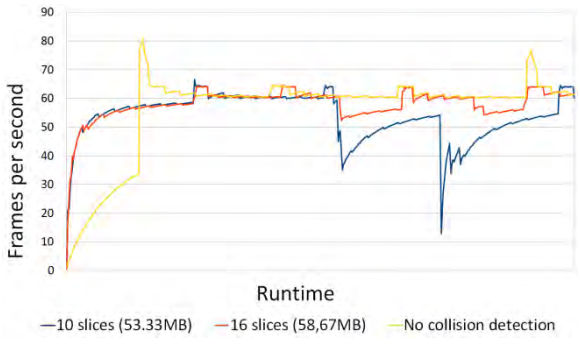


Figure 6. Average frame rate and memory consumption comparison between different n_{slices} configurations with 700x700 images, and no collision detection scenario. (Experiment B, Cathedral)

Results also show that our algorithm did not affect in any way the rendering speed of the interactive application. Figure 6 shows that the frame-rate was nearly identical in the same scenario with and without collision detection using 16 slices.

Although we did not aim for high precision on collision response, our technique has presented precise results on different resolutions. We note that point clouds are themselves approximations to surfaces, and as such a collision amongst points is an unlikely event, Figueiredo et. al use the average closest point to point distance divided by two to establish a conservative bounding box around each point, which can generate false collisions when close. With our approach, instead of a box, we use the pixels of a zoomed out view which is also conservative. Precision results with the different algorithms were verified empirically by changing the

Model	Time (s)	Total Complexity	Memory
Office	41,23	9.349.585 pts	610,3 MB
Church	58,14	6.475.125 pts	536,58 MB
Sibenik	78,87	5.199.093 pts	532,48 MB
Columns	42,92	2.612.303 pts	241,66 MB
Street	114,75	7.142.361 pts	598,02 MB
Entrance	13,9	580.062 pts.	122,88 MB
Room	6,96	271.731 pts.	67,58 MB

Table 2. Time, Total complexity (with generated points) and Memory consumption on the pre-processing stage for a 700x700

	Image-based (700x700)	BVH Oct 4096
Frame-rate	30 fps	16 to 30 fps
Total Memory	143.36 MB	225,44 MB
Pre. proc. time	13.9 s	1500 s

Table 3. Comparison between point-cloud techniques (Experiment A)

color of the avatar when a collision occurs. We found that using collision maps with a resolution of 700x700 enabled one to lower the number of false collisions from other methods when close to obstacles.

Floor collision has been performed perfectly in all resolutions, since its precision is defined by the rgb value of the point. Collisions with obstacles are more affected by resolution as is to be expected, since we rely on pixel finesse to precisely know the position of a given surface. Tests on office and street (Figure 7) have showed the same errors of small object interference or fake collisions due to diffuse information about object boundaries. These are more noticeable on the interactions with round objects on Street (Figure 7) where we can notice the aliasing creating invisible square barriers around a perfectly round object.

Table 3 compares our technique with the work from Figueiredo et. al [MJBJ], which has been tested on one of our experimental scenarios (Figure 7), the walkthrough in the point cloud of the Batalha Monastery (Experiment A in the accompanying video, 700x700 resolution n_{slices}set to 10), on a machine with a slightly faster processing speed and RAM than the one used for our walkthroughs. We compared our results with their best performance alternative, that bases the surface partition on 4096 cells of the octree.

Frame-rate was disturbed during the collision detection process on the R-tree approach, while it remained steady at the 30 fps during the whole execution of our application. Also, the image-based technique has required much less memory to be executed, even with a significantly high number of slices loaded in memory. The biggest difference is in the pre-processing times. Our approach was executed 107 times faster than the BVH approach. The pre-processing stage must only be performed once for each configuration, since the representation is written and loaded to the hard-drive for further interactions.

As stated in section 2 the research on point cloud collision detection is recent, and non-existent regarding image-based techniques. Our pioneer solution has presented excellent results, not only performing better than other works on point clouds published in the scientific community, but also being flexible enough to be applied on models from CAD, or combined with precise collision response techniques. Our technique can be performed with our representation on any computer that can render the input model at an acceptable frame-rate, without requiring much processing from the CPU or GPUs.

5. CONCLUSION AND FUTURE WORK

A new image-based environment representation for collision detection has been presented, using 2.5+D slices of an environment or buildings across the z axis. These images contain at each pixel, information about a given voxel, representing i's contents with colors. Height map information is stored on the red channel, and obstacles are indicated on the blue channel. This allows us to have a Broad phase collision detection stage that is performed with high efficiency and scalability, where the user can choose the precision according to the computing power at hand by simply adjusting the resolution of the produced images. Point clouds and polygonal models are ubiquitously processed, making our approach currently the best suited for fast interaction with massive laser-scan data. This work fills a gap in the area of collision detection, exploring a scenario that has been receiving more attention recently.

Future Work

Using graphic card capabilities such as the stencil buffer for broad-phase collision detection and vertex shaders for point coloring could greatly speed up these operations, and also, calculations would be moved to the GPU, taking away some work from the CPU. Applying this representation of environments also on objects of the scene, or even applying it to the avatars we're using on the interaction, could present interesting results. Using non-uniform resolution images on environments where we do not have a uniform complexity, would also help us achieve more precision on our narrow phase, or on these presented situations.

Image comparison techniques and compression can also be applied to the generated images in order to decrease the number of times we need to load a slice, and also the number of collision detection checks we must do. In several man-made structures such as buildings, many slices tend to be identical; intra-slice compression also presents itself as an interesting avenue of research.

6. ACKNOWLEDGMENTS

We would like to thank Artescan for the point clouds provided.

7. REFERENCES

[BHMTMG] Bruno Heidelberger, Matthias Teschner, and Markus Gross, Real-Time Volumetric Intersections of Deforming Objects, Proceedings of Vision, Modeling, and Visualization 2003,461-468,2003

[BMM] Brian Mirtich, V-clip: fast and robust polyhedral collision detection, ACM Trans. Graph., 17:177--208, July 1998.

[CLFTYC] Céline Loscos, Franco Tecchia, and Yiorgos Chrysanthou, Real-time shadows for animated crowds in virtual cities, In Proceedings of the ACM symposium on Virtual reality software and technology, VRST '01, pages 85--92, New York, NY, USA, 2001. ACM.

[DKDP] Dave Knott and Dinesh K. Pai, Cinder - collision and interference detection in real-time using graphics hardware, Proceedings of Graphics Interface, pages 73--80, 2003.

[FSJF] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou, Image-based collision detection and response between arbitrary volume objects, In Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08, pages 155--162, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[GBWHS] G. Baciu, Wingo Sai-Keung Wong, and Hanqiu Sun, Recode: an image-based collision detection algorithm, In Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on, pages 125 --133, oct 1998.

[GBWSW] George Baciu and Wingo Sai-Keung Wong, Hardware-assisted self collision for deformable surfaces, Proceedings of the ACM symposium on Virtual reality software and technology, 2002.

[JFHFCP] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry, Volume contact constraints at arbitrary resolution, ACM Trans. Graph., 29:82:1--82:10, July 2010.

[JKGZ] Jan Klein and Gabriel Zachmann, Point cloud collision detection, Computer Graphics Forum, 23(3):567--576, 2004.

[JSLR] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski, Layered depth images, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998.

[KMOOTK] Karol Myszokowski, Oleg G. Okunev, and Tosiyasu L. kunii, Fast collision detection between complex solids using rasterizing graphics hardware, The Visual Computer, 11(9):497 -- 512, 1995.

[MJBJ] Mauro Figueiredo, João Oliveira, Bruno Araújo, and João Pereira, An efficient collision detection algorithm for point cloud models, 20th International conference on Computer Graphics and Vision, 2010.

[MLJC] M.C. Lin and J.F. Canny, A fast algorithm for incremental distance calculation, In Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, pages 1008 --1014 vol.2, apr 1991.

[MT] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser, Collision detection for deformable objects, Computer Graphics Forum,24(1):61--81,2005

[NBJM] Niels Boldt and Jonas Meyer, Self-intersections with cullide, Eurographics, 23(3), 2005.

[NMAS] Noralizatul Azma Bt Mustapha Abdullah, Abdullah Bin Bade, and Sarudin Kari, A review of collision avoidance technique for crowd simulation, 2009 International Conference on Information and Multimedia Technology, (2004):388--392, 2009.

[NSRMLDM] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha, Cullide: interactive collision detection between complex models in large environments using graphics hardware, In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '03, pages 25--32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[PMH] Philip M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, ACM Transactions on Graphics, 15(3):179--210, July 1996.

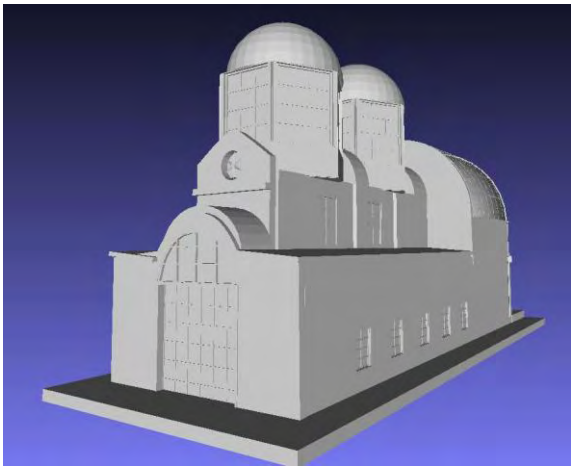
[SGMCLDM] S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: a hierarchical structure for rapid interference detection. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96). ACM, New York, NY, USA, 171-180.

[SKMNFF] Stephan Kimmerle, Matthieu Nesme, and François Faure, Hierarchy Accelerated Stochastic Collision Detection, In 9th International Workshop on Vision, Modeling, and Visualization, VMV 2004, pages 307-312, Stanford, California, États-Unis, November 2004.

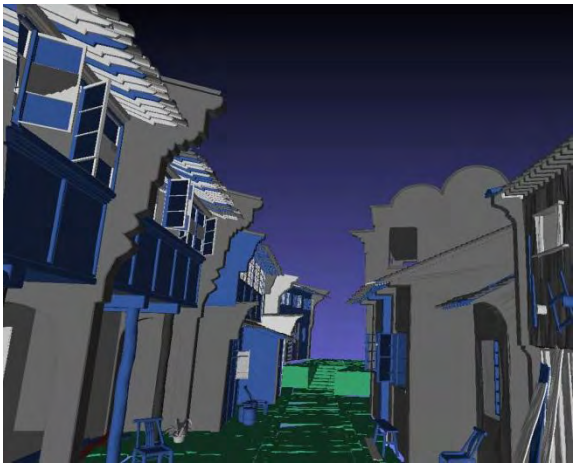
[SKTGKICB] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and Richard Rowe, Collision detection: A survey, In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pages 4046 --4051, oct. 2007.

[TLTAM] Thomas Larsson and Tomas Akenine-Möller, Collision detection for continuously deforming bodies, Eurographics 2001.

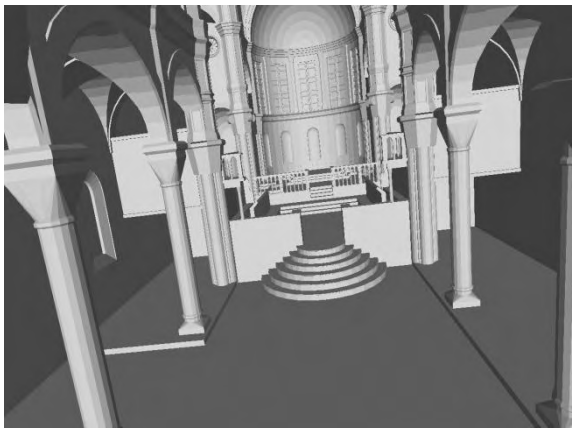
[XZYJK] Xinyu Zhang and Y.J. Kim, Interactive collision detection for deformable models using streaming aabbs, Visualization and Computer Graphics, IEEE Transactions on, 13(2):318 --329, march-april 2007.



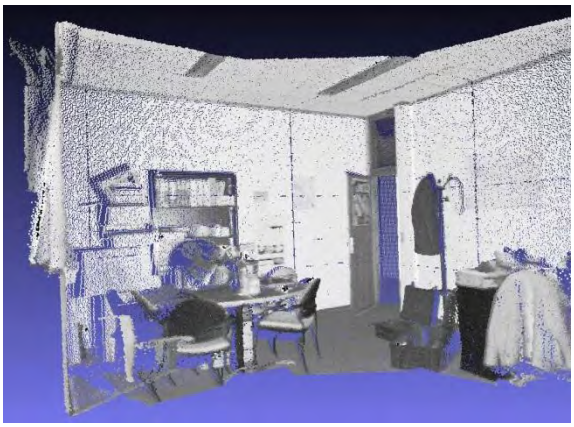
(a) Church



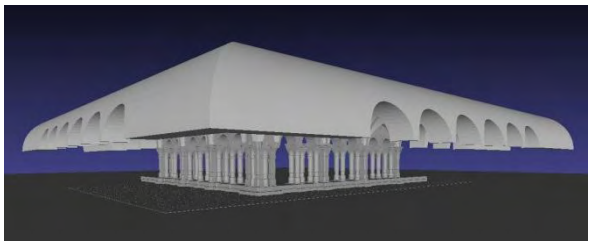
(b) Street



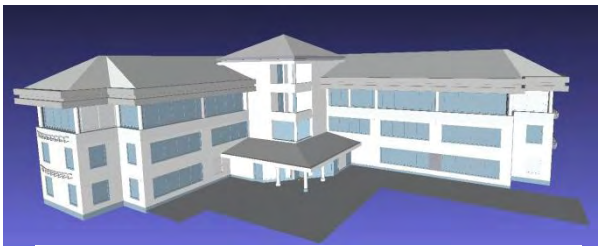
(c) Sibenik



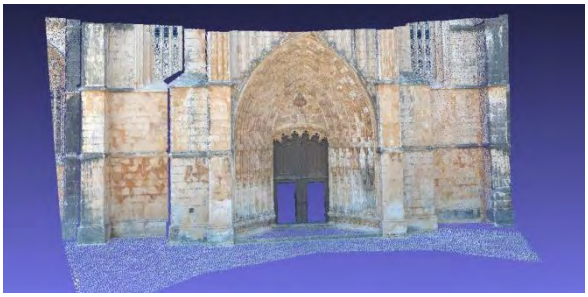
(d) Room



(e) Columns



(f) Office



(e) Batalha

Figure 7. Pictures of all the tested input polygonal models and point clouds. Environments with different topologies were chosen for this purpose.

Color Preserving HDR Fusion for Dynamic Scenes

Gökdeniz Karadağ
Middle East Technical University, Turkey
gokdeniz@ceng.metu.edu.tr

Ahmet Oğuz Akyüz
Middle East Technical University, Turkey
akyuz@ceng.metu.edu.tr

ABSTRACT

We present a novel algorithm to efficiently generate high quality high dynamic range (HDR) images. Our method is based on the idea of expanding the dynamic range of a reference image at granularity of tiles. In each tile, we use data from a single exposure, but different tiles can come from different exposures. We show that this approach is not only efficient and robust against camera and object movement, but also improves the color quality of the resulting HDR images. We compare our method against the commonly used HDR generation algorithms.

Keywords: High dynamic range imaging, image fusion, color quality

1 INTRODUCTION

The interest in HDR imaging has rapidly gained popularity in recent years. This has been accompanied by the development of various methods to create HDR images. While it is believed that using dedicated HDR capture hardware will be the de-facto way of generating HDR images in future [Rei10a], software solutions are still commonly used in today's systems. Among these multiple exposure techniques (MET) are the most dominant [Man95a, Deb97a].

In METs, several images of the same scene are captured by varying the exposure time between the images. This ensures that each part of the captured scene is properly exposed in at least one image. The individual images are then merged to obtain the HDR result. Although variations exist, the equation below is typically used for the merging process:

$$I_j = \sum_{i=1}^{N} \frac{f^{-1}(p_{ij})w(p_{ij})}{t_i} \Big/ \sum_{i=1}^{N} w(p_{ij}). \qquad (1)$$

Here N is the number of LDR images, p_{ij} is the value of pixel j in image i, f is the camera response function, t_i is the exposure time of image i, and w is a weighting function used to attenuate the contribution of poorly exposed pixels.

In Equation 1, a weighted average is computed for every pixel. While this may be desirable for attenuating noise, it introduces unwanted artifacts due to ghosting and misalignment problems. In this paper, we show that this approach also results in the desaturation of colors making the HDR image less saturated than the its constituent exposures.

Computing a weighted average for every pixel also requires that the individual pixels are perfectly aligned. Otherwise, pixels belonging to different regions in the scene will be accumulated resulting ghosting and alignment artifacts.

In this paper, we propose a method that largely avoids both of these problems. Our method is underpinned by the idea that instead of computing an average for every pixel, one can use the pixels from a single properly exposed image. A different image can be used for different regions ensuring that the full dynamic range is captured. We also introduce the concept of working in tiles instead of pixels to make the algorithm more robust against local object movements.

2 PREVIOUS WORK

Starting with the pioneering works of Madden [Mad93a] and Mann and Picard [Man95a], various algorithms have been developed to create HDR images. The early work focused on recovering the camera response function and choosing an appropriate weighting function [Deb97a, Mit99a, Rob03a, Gro04a]. These algorithms assumed that the exposures that are used to create an HDR image are perfectly aligned and the scene is static.

Ward developed a method based on median threshold bitmaps (MTBs) to allow photographers use hand-held images of static scenes in HDR image generation [War03a]. His alignment algorithm proved to be very successful and is used as an initial step of more advanced alignment and ghost removal algorithms [Gro06a, Jac08a, Lu09a].

In another alignment algorithm, Cerman and Hlaváč estimated the initial shift amounts by computing the correlation of the images in the Fourier domain [Cer06a]. This, together with the initial rotational estimate which

was assumed to be zero, was used as a starting point for the subsequent iterative search process.

Tomaszewska and Mantiuk employed a modified scale invariant feature transform (SIFT) [Low04a] to extract local features in the images to be aligned [Tom07a]. The prominent features are then selected by the RANSAC algorithm [Fis81a]. This refined set of features are then used to compute a homography between the input images.

Several methods have been proposed to deal with ghosting artifacts. These algorithms usually pre-align the input exposures using MTB or other algorithms to simplify the ghost detection process. Some of these algorithms avoid merging suspicious regions where there is high variance [Kha06a, Gal09a, Ram11a]. Other algorithms try to detect the movement of pixels and perform pixel-wise alignment [Zim11a]. A recent review of HDR ghost removal algorithms can be found in Srikantha and Sidibé [Sri12a].

There are also existing algorithms that attempt to combine data from multiple exposures for the purpose of generating a single low dynamic range (LDR) image. Among these, Goshtasby first partitions the images into tiles [Gos05a]. For each tile, he then selects the image that has the highest entropy. The tiles are blended using smooth blending functions to prevent seams. Mertens et al., on the other hand, do not use tiles but utilize three metrics namely contrast, saturation, and well-exposedness to choose the best image for each pixel [Mer07a]. Similar to Goshtasby, Várkonyi-Kóczy et al. propose a tile based algorithm where tiles are selected to maximize detail using image gradients [Var08a]. In another tile based algorithm, Vavilin and Jo use three metrics; mean intensity, intensity deviation, and entropy to choose the best exposure for each tile [Vav08a]. In contrast to previous tile based studies, they choose tile size adaptively based on local contrast. Finally, Jo and Vavilin propose a segmentation based algorithm which allows choosing different exposures for different clusters [Jo11a]. Unlike previous methods they use bilateral filtering during the blending stage.

It is important to note that existing tile-based algorithms attempt to generate LDR images with more details and enhanced texture information, whereas our goal is to generate HDR images with natural colors. Our approach alleviates the need for explicit ghost detection and removal procedures. If the dynamic parts of a scene do not span across regions with significantly different luminance levels, no ghost effects will occur in the output. Also, we avoid redundant blending of pixels that can result in reduced color saturation.

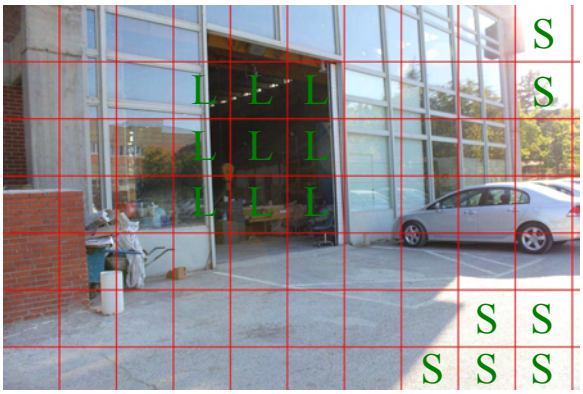


Figure 1: We partition the images into tiles and determine which exposure to use for each tile.

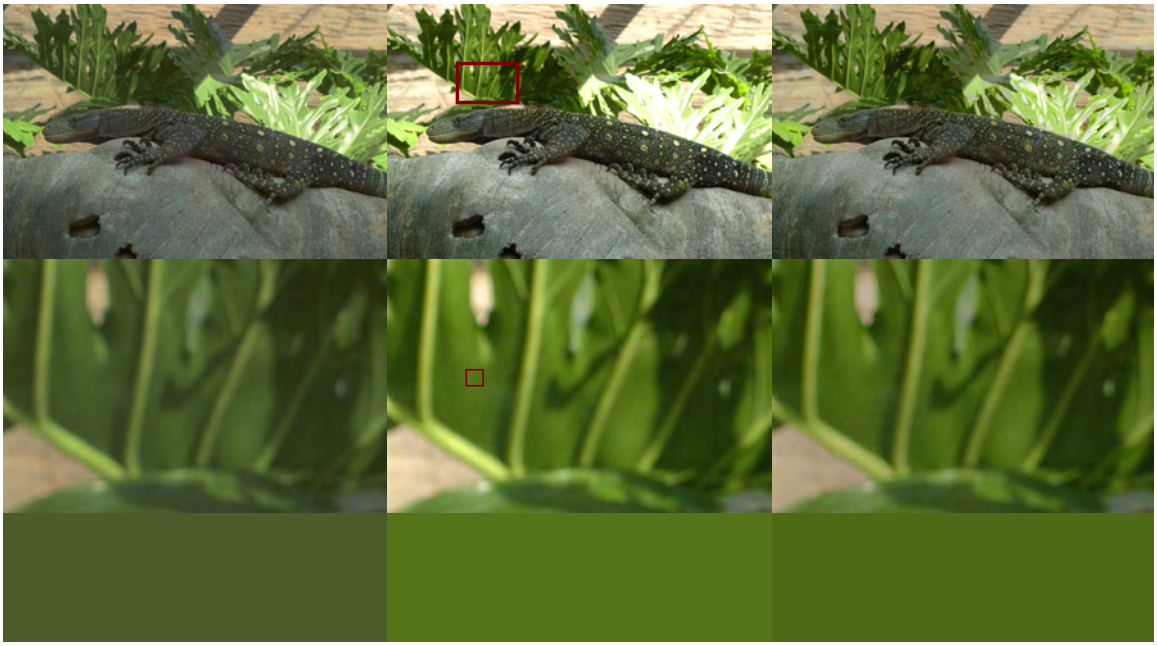
3 ALGORITHM

The first step of our algorithm is to align the input exposures using the MTB algorithm [War03a]. In this part, both the original MTB or the MTB with the rotation support can be used.

Once the images are aligned, we partition each exposure into tiles. Our goal then becomes to choose the best image that represents the area covered by each tile. A sample image is shown in Figure 1 to illustrate this idea. In this image, the under-exposed tiles are marked with **L** indicating that these tiles should come from a longer exposure. Similarly, over-exposed regions are marked by **S** suggesting that shorter exposures should be used for these tiles. Unmarked tiles can come from the middle exposures.

To make these decisions, we need to define a quality metric that indicates whether a tile is well-exposed. To this end, we experimented with the mean intensity as well as the number of under- and over-exposed pixels within a tile as potential metrics. Our results suggested that using the mean intensity gives better results. Therefore, we marked a tile as a *good* tile if its mean intensity is in the range $[I_{min}, I_{max}]$. I_{min} and I_{max} are user parameters, but we found that $I_{min} = 50$ and $I_{max} = 200$ can be used as reasonable defaults.

Based on this criteria, we compute the number of good tiles for each exposure. We choose the exposure with the maximum number of good tiles as the reference exposure. This exposure serves as the *donor* which provides data for all tiles whose mean intensity stays in the aforementioned limits. This leniency allows us to use the same image as much as possible and provides greater spatial coherency. For the remaining tiles, we choose the second reference exposure and fill in the tiles which are valid in this exposure. This process is



(a) Standard MET (b) LDR reference (c) Our result

Figure 2: (a) HDR image created by using the standard MET. (b) Selected individual exposure from the bracketed sequence. (c) HDR image created using our algorithm. The top row shows the full images. The middle row shows the close-up view of a selected region. The bottom row shows the color of a single pixel from the region indicated in the middle row. Both HDR images are tone mapped using the photographic tone mapping operator [Rei02a]. As can be seen in the zoomed views, the color quality of our result is closer to the selected reference image.

recursively executed until a source image is found for all tiles[1]. This process can be represented as:

$$I_j = \sum_{i=1}^{N} \frac{f^{-1}(p_{ij})W_{ij}}{t_i}, \tag{2}$$

$$W_{ij} = \begin{cases} 1 & \text{if pixel } j \text{ comes from image } i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that we no longer have the $w(p_{ij})$ term from Equation 1 as we do not compute a weighted average.

Finally, we use a blending strategy to prevent the visibility of seams at tile boundaries. For this purpose, we create Gaussian pyramids of weights (W_{ij}) and Laplacian pyramids of source images. We then merge the images by using Equation 2 at each level of the pyramid and collapse the pyramid to obtain the final HDR image. We refer the reader to Burt and Adelson's original paper for the details of this process [Bur83a].

Since the tiles are not overlapping our algorithm ensures that within each tile data from only a single source image is used. As we demonstrate in the next section, this improves the color saturation of the resulting HDR images. A second observation is that each tile is spatially coherent. This means that motion related artifacts will not occur within tiles. However, such artifacts can still occur across tiles. Thus our algorithm reduces the effect of motion artifacts but does not completely eliminate them.

[1] It is possible that the a tile is under- or over-exposed in all input images. In this case, we choose the longest exposure if the tile is under-exposed and shortest exposure otherwise.

4 RESULTS AND ANALYSIS

We present the results of our color preserving HDR fusion algorithm under three categories namely: (1) Fixed camera & static scene, (2) hand-held camera & static scene, and (3) hand-held camera & dynamic scene. For the first configuration, we illustrate that the color quality of the HDR image created by our method is superior to the output of the standard HDR fusion algorithm shown in Equation 1. A sample result for this case is depicted in Figure 2 where the output of the standard MET is shown on the left and our result is shown on the right. A selected exposure from the bracketed sequence is shown in the middle for reference.

For the image on the left, we used the tent weighting function proposed by Debevec and Malik [Deb97a]. We used the sRGB camera response function for both images, and a tile size of 64×64 for our result. It can be seen that, due to the pixel-wise averaging process, the output of the standard MET has a washed-out appearance. Our result, on the other hand, is colorimetrically closer to the selected exposure. This is a consequence of avoiding unnecessary blending between images.

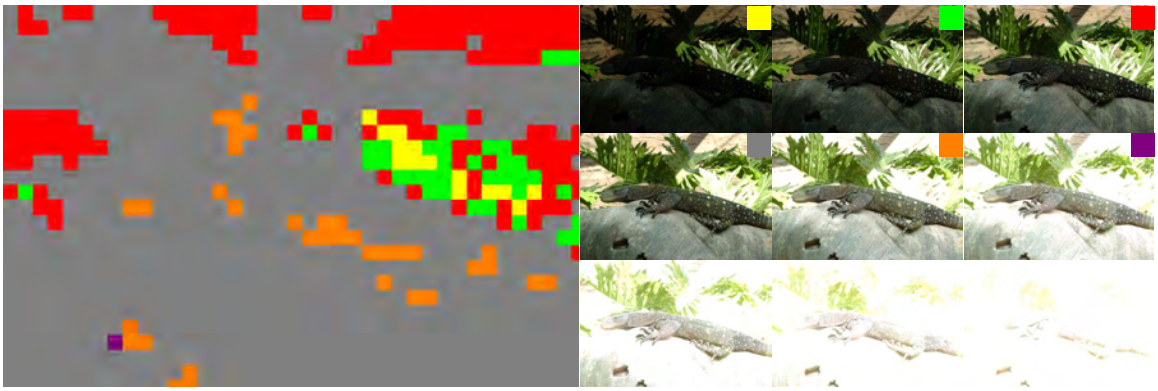


Figure 3: The colors show the correspondence between the tiles in the HDR image and the source images that they were selected from. We can see that most tiles were selected from the fourth image. Figure courtesy of Erik Reinhard [Rei10a].

Figure 3 shows which tiles in the output HDR image came from which images in the exposure sequence. The correspondence is shown by color coding the individual exposures. As we can see from this figure, the majority of the tiles were selected from the fourth exposure. The tiles that correspond to the highlights on the plants came from the darker exposures. On the other hand, the tiles that correspond to the crevices on the rock and the shadow of the lizard came from the lighter exposures. We can also see that the last three exposures were not used at all.

At this point, it would be worthwhile to discuss why the standard MET gives rise to a washed-out appearance and our algorithm does not. We would not expect to see this problem if all exposures were perfect representations of the actual scene. However, in reality, there are slight differences between exposures that are not only due to changing the exposure time. Slight camera movements, noise, and inaccuracies in the camera response curve can all cause variations between the actual observations. The combined effect of these variations result in reduced color saturation. By avoiding unnecessary blending, we also avoid this artifact.

The second test group consists of images of a static scene captured by a hand-held camera (Figure 4). In this figure, the left column shows the unaligned result created by directly merging five bracketed exposures. The middle column shows the tone mapped HDR output after the exposures are aligned by using the MTB algorithm. The right column shows our result obtained by first aligning the exposures using the MTB algorithm, and then merging them using our tile-based technique. As can be seen from the fence and the sign in the insets, our result is significantly sharper than that of the MTB algorithm. However, we also note that small artifacts are visible in our result on the letters “R” and “E”. Further examination reveals that these artifacts are due to using tiles from different exposures that are not perfectly aligned.

As the color map indicates, the majority of the final HDR image is retrieved from the exposure coded by red (exposures not shown). The darker regions retrieved data from the lighter (gray) exposure. The highlights at the top left corner received data from the darker (green) exposure. In fact, in this example, all five exposures contributed to the final image but the majority of the contribution came from these three exposures.

In the final category, we demonstrate the performance of our algorithm using scenes that have both global and local movement. To this end, we used the hdrgen software[2] which implements the MTB alignment algorithm and a variance based ghost removal method explained in Reinhard et al. [Rei10a]. In Figure 5, the left column shows the output obtained by only image alignment but without ghost removal. The middle column shows the result of alignment and ghost removal. Although the majority of the ghosts are removed, some artifacts are still visible on the flag as shown in the close-ups. The right column shows our result where these artifacts are eliminated. The color map indicates the source images for different regions of the HDR image.

We also demonstrate a case where our algorithm introduces some unwanted artifacts in high contrast and high frequency image regions as the window example in Figure 6. The bright back light and window grates cause high contrast. If the tile size is large, blending tiles from different exposures produces sub-par results. A reduced tile size eliminates these artifacts.

Our choice of prioritizing the reference image increases success in image sets where ghosting effects would normally occur. If the object movements are located in regions with similar lighting conditions, our algorithm prefers the image closer to reference image while constructing tiles, preventing ghosting effects. It is possible that an object moves between regions of different lighting conditions, and our algorithm may choose tiles

[2] `http://www.anyhere.com`

Figure 4: Left: Unaligned HDR image created from hand-held exposures. Middle: Exposures aligned using the MTB algorithm. Right: Our result. The close-ups demonstrate that our algorithm produces sharper images. The color map shows the source exposures for different regions of the HDR image.

(a) Alignment (b) Alignment and ghost removal (c) Our result

Figure 5: Left: Aligned HDR image created from hand-held exposures using the MTB algorithm. Middle: Aligned and ghost removed HDR image. Right: Our result. The insets demonstrate that ghosting artifacts are eliminated in our result. The color map shows the source exposures for different regions of the HDR image.

from different images where the moving object can be seen. In this case different copies of the object may be present in multiple locations in the output image.

Finally, we report the running times of our algorithm. An unoptimized C++ implementation of our algorithm was able to create high resolution (18 MPs) HDR images from 9 exposures within 30 seconds including all disk read and write times. We conducted all of our test on an Intel Core i7 CPU running at 3.20 GHz and equipped with 6 GBs of memory. This suggests that our algorithm is practical and can easily be integrated into existing HDRI workflows.

5 CONCLUSIONS

We presented a simple and efficient algorithm that improves the quality of HDR images created by using multiple exposures techniques. By not redundantly averaging pixels in low dynamic regions, our algorithm preserves the color saturation of the original exposures, and reduces the effect of ghosting and alignment artifacts. As future work, we are planning to make the tiling process adaptive instead of using a uniform grid. This would prevent artifacts that can be caused by sudden illumination changes between neighboring tiles coming from different exposures. We are also planning to perform blending using edge-aware Laplacian pyramid [Par11a] to avoid blending across sharp edges. Improved quality of our results can also be validated by a user study.

ACKNOWLEDGMENTS

This work was partially supported by METU BAP-08-11-2011-123.

6 REFERENCES

[Bur83a] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532 – 540, apr 1983.

Figure 6: Top: A tone-mapped HDR image with 128x128 tile size. Tile boundaries are highly visible in the close-up. Middle: Changing the tile size to 32x32 removes most of the artifacts, but some remain in diagonal lines. Bottom: Using a 2x2 tile size eliminates remaining artifacts.

[Cer06a] Lukáš Cerman and Václav Hlaváč. Exposure time estimation for high dynamic range imaging with hand held camera. In *Computer Vision Winter Workshop, Czech Republic*, 2006.

[Deb97a] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97 Conf. Proc.*, pages 369–378, August 1997.

[Fis81a] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[Gal09a] O. Gallo, N. Gelfandz, Wei-Chao Chen, M. Tico, and K. Pulli. Artifact-free high dynamic range imaging. In *Computational Photography (ICCP), 2009 IEEE International Conference on*, pages 1 –7, april 2009.

[Gos05a] A. Ardeshir Goshtasby. Fusion of multi-exposure images. *Image and Vision Computing*, 23(6):611 – 618, 2005.

[Gro04a] M.D. Grossberg and S.K. Nayar. Modeling the space of camera response functions. *Pattern Analysis and Machine Intelligence, IEEE Trans. on*, 26(10):1272 –1282, 2004.

[Gro06a] Thorsten Grosch. Fast and robust high dynamic range image generation with camera and object movement. In *Proc. of Vision Modeling and Visualization*, pages 277–284, 2006.

[Jac08a] Katrien Jacobs, Celine Loscos, and Greg Ward. Automatic high-dynamic range image generation for dynamic scenes. *IEEE CG&A*, 28(2):84–93, 2008.

[Jo11a] K.H. Jo and A. Vavilin. Hdr image generation based on intensity clustering and local feature analysis. *Computers in Human Behavior*, 27(5):1507–1511, 2011.

[Kha06a] Erum Arif Khan, Ahmet Oğuz Akyüz, and Erik Reinhard. Ghost removal in high dynamic range images. *IEEE International Conference on Image Processing*, 2006.

[Low04a] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[Lu09a] Pei-Ying Lu, Tz-Huan Huang, Meng-Sung Wu, Yi-Ting Cheng, and Yung-Yu Chuang. High dynamic range image reconstruction from hand-held cameras. In *CVPR*, pages 509–516, 2009.

[Mad93a] B. C. Madden. Extended dynamic range imaging. Technical report, GRASP Laboratory, Uni. of Pennsylvania, 1993.

[Man95a] S Mann and R Picard. Being 'undigital' with digital cameras: Extending dynamic range by combining differently exposed pictures, 1995.

[Mer07a] T. Mertens, J. Kautz, and F. Van Reeth. Exposure fusion. In *Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on*, pages 382 –390, 29 2007-nov. 2 2007.

[Mit99a] T. Mitsunaga and S. K. Nayar. Radiometric self calibration. In *Proceedings of CVPR*, volume 2, pages 374–380, June 1999.

[Par11a] Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local laplacian filters: edge-aware image processing with a laplacian pyramid. *ACM Trans. Graph.*, 30(4):68:1–68:12, July 2011.

[Ram11a] Shanmuganathan Raman and Subhasis Chaudhuri. Reconstruction of high contrast images for dynamic scenes. *The Visual Computer*, 27(12):1099–1114, 2011.

[Rei02a] Erik Reinhard, Michael Stark, Peter Shirley, and Jim Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics*, 21(3):267–276, 2002.

[Rei10a] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann, San Francisco, second edition edition, 2010.

[Rob03a] Mark A. Robertson, Sean Borman, and Robert L. Stevenson. Estimation-theoretic approach to dynamic range enhancement using multiple exposures. *Journal of Electronic Imaging 12(2), 219 228 (April 2003).*, 12(2):219–228, 2003.

[Sri12a] Abhilash Srikantha and Désiré Sidibé. Ghost detection and removal for high dynamic range images: Recent advances. *Signal Processing: Image Communication*, (0):–, 2012.

[Tom07a] Anna Tomaszewska and Radoslaw Mantiuk. Image registration for multi-exposure high dynamic range image acquisition. In *WSCG: Proc. of the 15th Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2007.

[Var08a] A. R. Varkonyi Koczy, A. Rovid, and T. Hashimoto. Gradient-based synthesized multiple exposure time color hdr image. *Instrumentation and Measurement, IEEE Transactions on*, 57(8):1779 –1785, aug. 2008.

[Vav08a] A. Vavilin and K.H. Jo. Recursive hdr image generation from differently exposed images based on local image properties. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 2791–2796. IEEE, 2008.

[War03a] Greg Ward. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of Graphics Tools*, 8(2):17–30, 2003.

[Zim11a] Henning Zimmer, Andrés Bruhn, and Joachim Weickert. Freehand hdr imaging of moving scenes with simultaneous resolution enhancement. *Computer Graphics Forum*, 30(2):405–414, 2011.