ISSN 1213-6972 *Volume 19, Number 1-3, 2011*

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, Univerzitni 8
306 14 Plzen
Czech Republic
skala@kiv.zcu.cz

Managing Editor: Vaclav Skala

Published and printed by 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*

ISBN 978-80-86943-84-8

Journal of WSCG

Editor-in-Chief

Vaclav Skala, University of West Bohemia
Centre for Computer Graphics and Visualization
Univerzitni 8, CZ 306 14 Plzen, Czech Republic
skala@kiv.zcu.cz http://herakles.zcu.cz
Journal of WSCG: URL: http://wscg.zcu.cz/jwscg

Direct Tel. +420-37-763-2473
Direct Fax. +420-37-763-2457
Fax Department: +420-37-763-2402

Editorial Advisory Board
MEMBERS

WSCG 2011

Board of Reviewers

Akleman, E. (United States)
Ariu, D. (Italy)
Assarsson, U. (Sweden)
Aveneau, L. (France)
Balcisoy, S. (Turkey)
Battiato, S. (Italy)
Benes, B. (United States)
Benoit, C. (France)
Biasotti, S. (Italy)
Bilbao, J. (Spain)
Biri, V. (France)
Bittner, J. (Czech Republic)
Bosch, C. (France)
Bouatouch, K. (France)
Boukaz, S. (France)
Bouville, C. (France)
Bruni, V. (Italy)
Buehler, K. (Austria)
Cakmak, H. (Germany)
Camahort, E. (Spain)
Capek, M. (Czech Republic)
CarmenJuan-Lizandra, M. (Spain)
Casciola, G. (Italy)
Coquillart, S. (France)
Correa, C. (United States)
Cosker, D. (United Kingdom)
Daniel, M. (France)
de Amicis, r. (Italy)
de Geus, K. (Brazil)
Debelov, V. (Russia)
Domonkos, B. (Hungary)
Drechsler, K. (Germany)
Duke, D. (United Kingdom)
Dupont, F. (France)
Durikovic, R. (Slovakia)
Eisemann, M. (Germany)
Erbacher, R. (United States)
Erleben, K. (Denmark)
Farrugia, J. (France)
Feito, F. (Spain)
Ferguson, S. (United Kingdom)
Fernandes, A. (Portugal)
Flaquer, J. (Spain)
Fontana, M. (Italy)
Fuenfzig, C. (France)
Gallo, G. (Italy)
Galo, M. (Brazil)
Garcia Hernandez, R. (Spain)
Garcia-Alonso, A. (Spain)
Gavrilova, M. (Canada)
Giannini, F. (Italy)
Gonzalez, P. (Spain)
Grau, S. (Spain)
Gudukbay, U. (Turkey)
Guggeri, F. (Italy)
Gutierrez, D. (Spain)
Habel, R. (Austria)
Hall, P. (United Kingdom)
Hansford, D. (United States)
Haro, A. (United States)
Hasler, N. (New Zealand)
Havemann, S. (Austria)
Havran, V. (Czech Republic)
Hernandez, B. (Mexico)
Herout, A. (Czech Republic)
Horain, P. (France)
House, D. (United States)
Chaine, R. (France)

Chaudhuri, D. (India)
Chmielewski, L. (Poland)
Chover, M. (Spain)
Iwasaki, K. (Japan)
Jansen, F. (Netherlands)
Jeschke, S. (Austria)
Jones, M. (United Kingdom)
Jones, M. (United States)
Juettler, B. (Austria)
Kheddar, A. (Japan)
Kim, H. (Korea)
Klosowski, J. (United States)
Kohout, J. (Czech Republic)
Kurillo, G. (United States)
Kyratzi, S. (Greece)
Lanquetin, S. (France)
Lay Herrera, T. (Germany)
Lee, T. (Taiwan)
Lee, S. (Korea)
Leitao, M. (Portugal)
Liu, D. (Taiwan)
Liu, S. (China)
Lutteroth, C. (New Zealand)
Madeiras Pereira, J. (Portugal)
Maierhofer, S. (Austria)
Manzke, M. (Ireland)
Marras, S. (Italy)
Maslov, O. (Russia)
Matey, L. (Spain)
Matkovic, K. (Austria)
Max, N. (United States)
Meng, W. (China)
Mestre, D. (France)
Michoud, B. (France)
Mokhtari, M. (Canada)
Molla Vaya, R. (Spain)
Montrucchio, B. (Italy)
Muehler, K. (Germany)
Murtagh, F. (Ireland)
Nishio, K. (Japan)
OliveiraJunior, P. (Brazil)
Oyarzun Laura, C. (Germany)
Pan, R. (China)
Papaioannou, G. (Greece)
Pasko, A. (United Kingdom)
Pasko, G. (Cyprus)
Patane, G. (Italy)
Patow, G. (Spain)
Pedrini, H. (Brazil)
Peters, J. (United States)
Pina, J. (Spain)
Platis, N. (Greece)
Puig, A. (Spain)
Puppo, E. (Italy)
Purgathofer, W. (Austria)
Reshetov, A. (United States)
Richardson, J. (United States)
Richir, S. (France)
Rojas-Sola, J. (Spain)
Rokita, P. (Poland)
Rosenhahn, B. (Germany)
Rudomin, I. (Mexico)
Sakas, G. (Germany)
Salvetti, O. (Italy)
Sanna, A. (Italy)
Segura, R. (Spain)
Sellent, A. (Germany)
Shesh, A. (United States)
Schultz, T. (United States)
Schumann, H. (Germany)
Sirakov, N. (United States)
Skala, V. (Czech Republic)
Slavik, P. (Czech Republic)
Sochor, J. (Czech Republic)
Sousa, A. (Portugal)
Srubar, S. (Czech Republic)
Stroud, I. (Switzerland)
Subsol, G. (France)
Sundstedt, V. (Sweden)
Tang, M. (China)
Tavares, J. (Portugal)
Teschner, M. (Germany)
Theoharis, T. (Greece)
Theussl, T. (Saudi Arabia)
Tokuta, A. (United States)
Tomori, Z. (Slovakia)
Torrens, F. (Spain)
Trapp, M. (Germany)

Journal of WSCG

Contents

Vol. 19, No.1-3

Robust Feature Point Matching in General Multi-Image Setups

Anita Sellent
TU Braunschweig, Germany
sellent@cg.tu-bs.de

Martin Eisemann
TU Braunschweig, Germany
eisemann@cg.tu-bs.de

Marcus Magnor
TU Braunschweig, Germany
magnor@cg.tu-bs.de

ABSTRACT

We present a robust feature matching approach that considers features from more than two images during matching. Traditionally, corners or feature points are matched between *pairs* of images. Starting from one image, corresponding features are searched in the other image. Yet, often this two-image matching is only a subproblem and actually robust matches over multiple views and/ or images acquired at several instants in time are required. In our feature matching approach we consider the multi-view video data modality and find matches that are consistent in three images. Requiring neither calibrated nor synchronized cameras, we are able to reduce the percentage of wrongly matched features considerably. We evaluate the approach for different feature detectors and their natural descriptors and show an application of our improved matching approach for optical flow calculation on unsynchronized stereo sequences.

Keywords: Keypoint matching, motion estimation, multi-view video.

1 INTRODUCTION

In recent years the increased availability of high quality video cameras together with readily available storage space and fast data transfer has led to a growing interest in stereoscopic or, more general, multiple view video. Although multi-view video data actually is highly redundant, many algorithms in the processing pipeline consider only pairs of images. One important processing step is establishing feature point correspondences that are used, e.g. as low-level starting point for motion estimation [SLW+10, BWSS09, BBM09]. Determination of robust feature points and corresponding feature point descriptions has been an intensely investigated area of research for decades [MTS+05, MS05]. In spite of great advances, wrongly matched correspondences are still commonly encountered. If additional information on the images is provided, e.g. by calibration, synchronization or assumption of constant rigid motion, this information can be used to eliminate wrongly matched correspondences [HZ03]. Unfortunately, in practical applications additional information is not always available as, for instance, multiple cameras are hard to synchronize in an outdoor environment and usually images of independently moving objects are recorded.

The goal of our work is to develop a versatile, robust feature point matching method that is generally applicable, e.g. also in the unconstrained multi-view video setup. Our basic idea is to exploit the redundancy in the data of multi-view video sequences with a common field of view. We use it to establish more reliable correspondences to ensure high-quality matches. Feature points are matched by considering loops of images. We introduce *three image consistent matching* and evaluate it by means of the percentage of wrong matches.

Additionally, we show how a stereo-video optical flow algorithm [SLM10] can benefit from incorporating our robustly matched features. Recent research has shown that optical flow can be improved if ideas from feature matching are included into the approach, [BBM09, XJM10]. In contrast to variational based optical flow algorithms that require an iterative approach to cope with large distances [BBPW04], features can be matched independently from their position in the image and thus deal with arbitrary distances, as long as their descriptor is sufficiently robust to the corresponding changes in perspective or object deformations. For the inclusion of feature matching, optical flow approaches pay careful attention to outier matches as these are able to prevent convergence to the desired motion fields. In this work we show that our robust loop matching strategy which exploits the data modality given for multi-view video is able to improve optical flow estimations without further outlier treatment.

2 RELATED WORK

Usually features are matched between two images from synchronized cameras and spurious matches can be discarded using epipolar geometry [SZ02, HZ03]. Generally, the assumption of global affine motion between

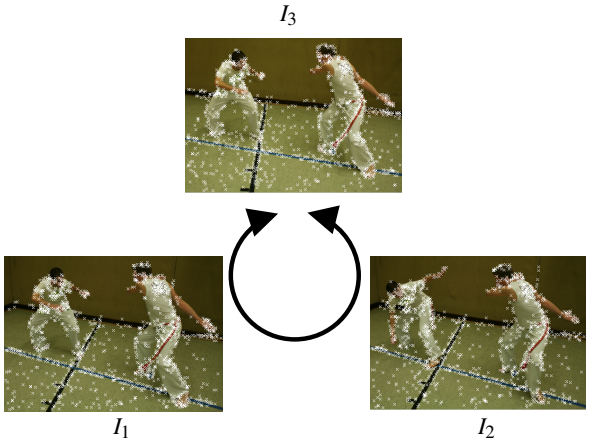


Figure 1: Three images with detected features (SIFT) of a multi-view video sequence: our algorithm accepts three images with some common field of view acquired by one or several unsynchronized and uncalibrated cameras. By requiring consistency of matches on a loop of three images, false matches are eliminated and correspondences between images can be established robustly.

two images can be used to validate matches [BGPS07]. But also game theoretic approaches exploiting local similarity transforms are used to establish reliable matchings between two images [ART10].

If several independent objects move in a monocular sequence, e.g. for person or object tracking [YJS06], feature locations from previous frames can also be used to estimate feature locations in the current frame [Zha94]. Assuming that features have at most one correct match in each frame, disjoint tracks of features over multiple frames can be considered to improve correspondences [VRB03, SS05, SSS06]. Thereby, the tracks provide a regularization of the matches over time, but no feedback for the correctness of the tracking is provided.

For static scenes, the trifocal tensor [TZ97] can be used to consider consistency of the matching between more than two images [BTZ96]. Yao and Cham first verify and add matches between image pairs to satisfy the epipolar constraint, before the matches are extended to image triples and the trifocal tensor is computed [YC07]. In contrast, Zach et al. first determine global, invertible transformations between image pairs before they detect wrong transformations on multi-image loops and discard them [ZKP10], enabling more robust multi-image static 3D reconstruction.

If a dynamic scene is recorded by multiple, unsynchronized cameras Ho and Pong work with high density feature points and use assignments of neighboring pixel in a relaxation labeling framework to obtain consistent matchings [HP96]. In the same setup, Ferrari et al. perform consistency checks on loops of images, but require an additional similarity measure that is different from the measure used to establish preliminary matchings [FTV03].

Mathematically the problem of finding consistent correspondences on three sets of equal, finite cardinality is well studied [Spi00] and approximation algorithms to the NP-hard problem have been proposed by several authors [CS92, BCS94].

In Sect. 3 we will adapt these approximation schemes to sets of different sizes. In Sect. 4 we evaluate the results of this new algorithm. We incorporate our consistent matches into a three image spatio-temporal optical flow algorithm, Sect. 5 and show how consistency of flow and features can improve dense correspondence estimation.

3 THREE IMAGE-FEATURE MATCHING

Let $I_1 : \Omega_1 \rightarrow \mathbb{R}$, $I_2 : \Omega_2 \rightarrow \mathbb{R}$ and $I_3 : \Omega_3 \rightarrow \mathbb{R}$ be three images of a multi-view video sequence that have some common field of view on a dynamic scene. In contrast to previous robust matching methods, we do not require epipolar geometry between images to be applicable, nor do we assume a temporal ordering, i.e. the three images can be acquired by one, two or three unsynchronized cameras, Fig. 1. For each image I_i, $i \in \{1,2,3\}$ a feature detector determines features $f_{i,k}, k \in \{1,\ldots,N_i\}$ with corresponding descriptors $s_{i,k}$. We denote the descriptor distance function with $d(s_{i,k}, s_{j,m})$. In our experiments, Sect. 4, we evaluate the algorithm for several detector/ descriptor variants, so we keep the description general in this section.

Usually, after detection the features are matched between two images at a time. Authors of different descriptors propose slightly different matching methods. To keep the results comparable, we follow the approach of [MS05] and use nearest neighbor matching (NN) for all two-matching steps.

A more elaborate two-matching strategy (NNDR) compares the distance of the nearest neighbor to the distance of the second nearest neighbor and only accepts a match if their ratio is below a threshold [Low04]. We also include this matching strategy into our evaluation.

If more than two images are considered, inconsistencies in the matches such as $(f_{1,k}, f_{2,m})$, $(f_{1,k}, f_{3,n})$ and $(f_{2,m}, f_{3,p})$, $p \neq n$ become obvious. In multi-view video, corresponding feature points are supposed to belong to one single scene point, so inconsistent matches indicate false matches. A straightforward approach to reduce the number of false matches is to filter out any match that is not consistent on a three image circle. To eliminate inconsistent matches already during the assignment we formulate the matching problem in a different way.

In our approach we look for *triples* $(f_{1,k}, f_{2,m}, f_{3,n})$ such that each $f_{i,j}$ is present in at most one triple. To each of the triples we assign a cost $\tilde{d}$ that is the sum of

the distances of all three descriptors $\tilde{d}(s_{1,k}, s_{2,m}, s_{3,n}) = d(s_{1,k}, s_{2,m}) + d(s_{1,k}, s_{3,n}) + d(s_{2,m}, s_{3,n})$, i.e. the distance between each pair of features is considered in the cost function, which therefore is independent of the ordering of the images. In contrast to previous approaches this formulation requires the matches in all images to be similar and thus closes the loop between the images, providing a feedback to the matching and avoiding the drift commonly encountered in considering ordered set of images. If all features were present in all three images this is an instance of the classical three-matching problem with decomposable cost-function, a NP hard problem which can be solved approximately with the following algorithm [CS92]:

i. Match the features in I_1 and I_2, e.g. using the Hungarian algorithm, (see [PS98]).

ii. Merge the sets of features on the basis of the matching in (i.) such that the new cost function between features in I_1 and I_3 is $\hat{d}(s_{1,k}, s_{3,n}) = \tilde{d}(s_{1,k}, s_{2,m}, s_{3,n})$.

iii. Match the features in I_1 and I_3 with the new distance function.

iv. Sum up all distances present in the matching.

v. Interchange the role of I_1, I_2, I_3 and restart at (i.).

vi. Of the three matchings thus obtained, return the one with the smallest sum of distances.

Note that step (ii.) enforces the third feature in the triple to be close both to the feature in I_1 and the feature in I_2. Enforcing this condition simultaneously provides the means to transport the information of the other images to the bilateral matching.

The three-match returned by this algorithm can be proved to lie within a certain distance to the actual best solution and in practice it often turns out to be the best solution [BCS94].

Yet, working with real images, we have to deal with occluded and non-detected features as well as with non-distinctive descriptors. We therefore adjust the above algorithm. In step (i.) we use NN matching or optionally NNDR matching. Additionally we match feature points only if they are mutual nearest neighbors. Thus the processing is independent from the ordering of the images and the feature points. For step (ii.) we remove all features from both images that are not matched in the previous step. We are only interested in feature points that can be matched consistently in three images. As the number of feature points differ in every image and we do not require all feature points to be matched, the sum of all matchings is no longer a reliable quality measure and step (iv.) is skipped. Correspondingly, for step (vi.) we do not return the match with the smallest overall cost, as this is dependent on the number of feature points actually matched. Instead we merge the three matches and only return those triples that are found in all three matching directions. Though this last step might seem rather restrictive, in our setup we opt for less matches with high quality instead of a higher number of matches with more questionable quality. This proceeding is in accordance with considering $\tilde{d}$ in (iii.) that enforces the matches to be mutual neighbors. In summary our algorithm looks as follows:

1. (a) Match the features in I_1 and I_2, using NN matching, optionally with distance check to the second nearest neighbor.
   (b) Match the features in I_2 and I_1, using NN matching, optionally with distance check to the second nearest neighbor.
   (c) Accept only symmetrically matched features.
2. Remove unmatched features in I_1 and merge the remaining features on the basis of the matching in (1.) such that the new cost function between matched features in I_1 and features in I_3 is $\hat{d}(s_{1,k}, s_{3,n}) = \tilde{d}(s_{1,k}, s_{2,m}, s_{3,n})$.
3. (a) Match the features in I_1 and I_3 with the new distance function using NN matching.
   (b) Match the features in I_3 and I_1 with the new distance function using NN matching.
   (c) Accept only symmetrically matched features.
4. Interchange the role of I_1, I_2, I_3 and restart at (1.).
5. Merge the three matchings and return only those matches that are assigned in all three matching directions.

4 EVALUATION OF THREE IMAGE-FEATURE MATCHING

A great number of feature detectors [MTS+05] and feature descriptors [MS05] exist in literature. For a comparison of those we refer the reader to these surveys. The aim of our work is to evaluate the impact of three-image matching and so we chose four widely used detector/ descriptor combinations for our evaluations: SIFT [Low04] and SURF [BETV08] are both scale invariant detectors for blob-like structures and with their natural descriptors also invariant to rotation and changes in illumination. We also evaluate our matching algorithm on Harris-corners [HS88] and the more recent accelerated corner detector FAST [RD06] and combine both with the normalized cross correlation (*NCC*) on a 9×9 window. We transform the normalized cross-correlation to a cost function via $d(s_{i,k}, s_{j,m}) = 1 - NCC(f_{i,k}, f_{j,m})$ to obtain a descriptor distance as used in Sect. 3. Using rather advanced and robust detectors as well as rather low level detectors we

		SIFT NN		SURF NN		Harris NN		FAST NN		SIFT NNDR	
		# M	%WM	# M	%WM	# M	%WM	# M	%WM	# M	%WM
art	2IM	1444	53.39	616	64.45	93	49.46	474	45.57	674	10.53
	3IM	603	11.28	177	20.90	44	13.64	220	13.64	506	2.57
books	2IM	1786	15.58	713	38.85	364	21.98	914	27.02	1506	2.52
	3IM	1373	2.26	318	8.81	200	9.00	517	8.70	1315	0.84
dolls	2IM	2206	23.75	809	35.60	134	18.66	812	19.33	1583	2.21
	3IM	1545	4.27	434	7.60	102	2.94	528	4.17	1367	1.02
laundry	2IM	1112	49.64	675	68.89	158	80.38	420	55.58	627	19.94
	3IM	550	15.82	193	28.50	32	40.63	174	17.24	457	7.66
moebius	2IM	1634	24.24	475	38.95	77	20.78	317	35.65	1211	4.54
	3IM	115	5.02	254	14.96	50	4.00	160	6.88	1011	2.47
reindeer	2IM	943	27.78	428	43.69	49	20.78	290	33.79	683	6.88
	3IM	664	7.08	200	14.50	37	8.11	143	11.89	578	2.77
waving	2IM	4345	11.12	1314	24.20	196	26.53	353	19.97	3804	1.26
	3IM	3995	4.76	1069	12.16	156	19.23	135	9.43	3720	0.70
stonemill	2IM	628	34.71	251	62.55	225	49.78	763	49.15	366	2.73
	3IM	427	13.11	114	35.96	133	27.82	452	22.79	324	0.62
RubberW.	2IM	2077	3.85	236	16.53	48	0.00	255	6.67	1975	0.56
	3IM	1585	0.32	107	5.61	25	0.00	153	1.31	1510	0.20
Hydr.	2IM	1111	16.56	432	20.88	176	25.57	576	22.74	853	1.52
	3IM	254	2.76	56	8.93	20	15.00	70	8.57	136	0.74
wall	2IM	7776	25.44	2365	49.26	1693	28.53	6733	33.71	5327	0.56
	3IM	5363	2.50	686	5.10	906	1.21	2892	1.87	4714	0.19
graffiti	2IM	2057	62.52	1385	77.98	265	90.68	822	91.12	689	25.83
	3IM	626	11.50	140	33.57	8	87.50	39	78.95	338	4.14

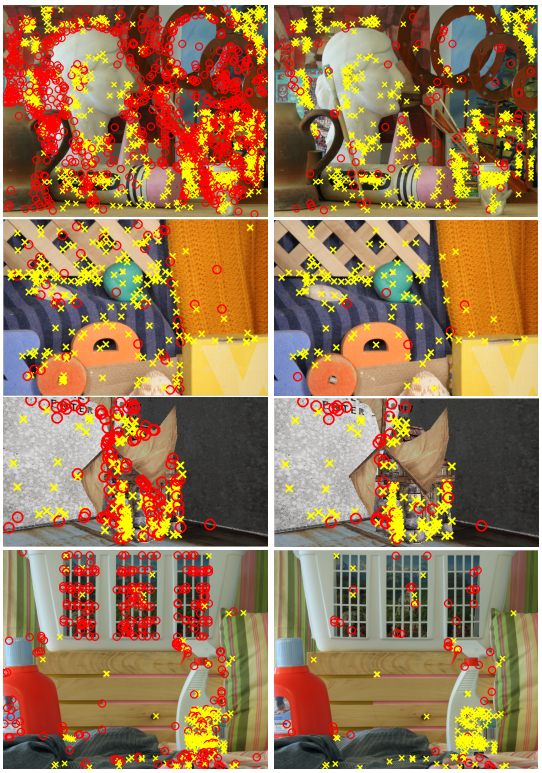
Table 1: As three image matches (3IM) have to satisfy stricter requirements than two image matches (2IM), the total number of matches is reduced while the quality of the matching is increased as the percentage of wrong matches (% WM) is considerably decreased no matter which of the feature detectors (SIFT, SURF, Harris or FAST) or matching strategy (nearest neighbor(NN) or nearest neighbor with threshold on the distance ratio (NNDR)) is used.

want to evaluate our matching scheme independently from the detector used.

For reason of comparison, in our experiments we apply nearest neighbor (NN) matching in all cases [MS05]. Additionally we apply the more advanced NNDR matching that was proposed for SIFT-features, using a threshold of 0.8 on the distance ratio [Low04]. We apply the thresholding step accordingly in the matching step (1.), but found it to have no impact in the matching step (3.) as the combined matching already is sufficiently distinguishing. We therefore do not apply the distance check in (3.).

Using a naïve MATLAB implementation on a 2.66GHz processor, three image consistent matching of 975 FAST features with 81 dimensional descriptors in I_1, 944 features in I_2 and 860 features in I_3 for the *art* scene requires 736ms. With the same setup, independent two-matchings between I_1 and I_2, I_1 and I_3 and I_2 and I_3 last together 126ms.

In our experiments we determine the number of matches and the percentage of matches outside a 5 pixel circle around the ground-truth location in different scenes. The scenes *art*, *books*, *dolls*, *laundry*, *moebius* and *reindeer* are rectified multiple view images of a static scene with known disparity [SP07]. The scenes *waving* [SLM10] and *stonemill* [LLM10] are synthetic, unsynchronized stereo sequences of a moving scene with known ground-truth correspondence fields. The scenes *RubberWhale* and *Hydrangea* are the only monocular sequences of more than two



(a) (b)

Figure 2: The two image-based matching approach (a) results in more outliers (red circles) and a lower relative amount of inliers (yellow crosses) than our three image based-matching (b). From top to bottom: scene *art* with SIFT features, *RubberWhale* with SURF features, *stonemill* with Harris corners, *laundry* with FAST features, all using nearest neighbor matching.

images with independently moving objects and known ground-truth motion from the Middlebury optical flow data set [BSL+07]. In contrast, the scenes *wall* and *graffiti* describe a viewpoint change for a static, mostly planar scene [MS05]. The number of matches and percentage of outliers are shown in Tab. 1, some examples are given in Fig. 2. As expected the number of matches is reduced with our stricter three-matching strategy. But at the same time the percentage of outliers among the assigned matches is also considerably reduced.

We also apply our algorithm to the real multi-video recordings scenes *market*, 421×452 pixel, and *capoeira*, 817×578 pixel, which are recorded using unsynchronized, uncalibrated cameras with automatic gain, while in the scene *outside*, 270×480 pixel, cameras are additionally hand-held. The algorithm is performed on the entire images with all features points found, but for visibility reasons, Fig. 3 shows the results only for 100 randomly selected SIFT-features: matched features are marked with a white x and connected via a yellow line to the location of the corresponding

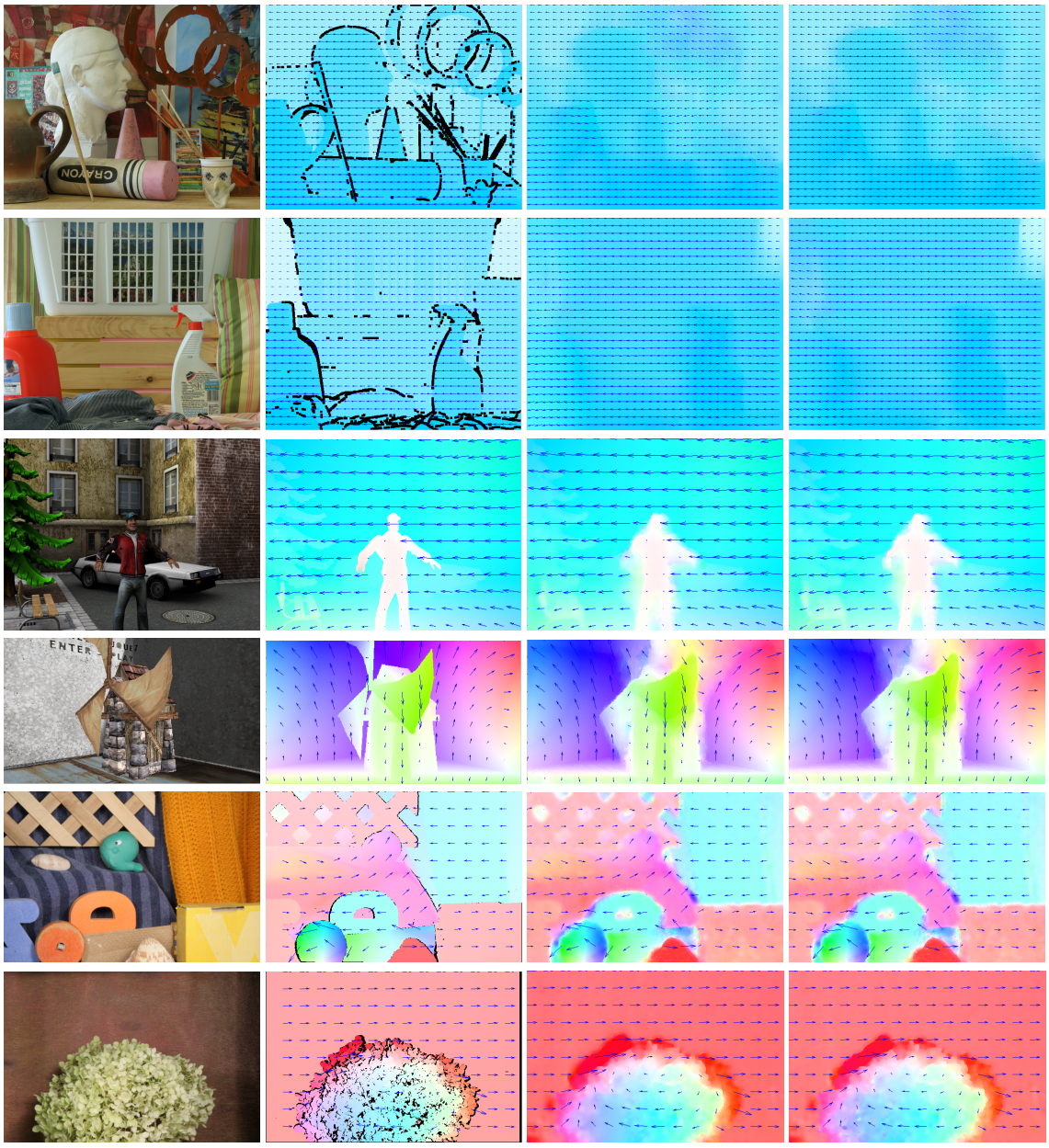
(a) (b) (c)

Figure 3: For three real world scenes *market*, *capoeira* and *outside* (a) we compare different matching strategies. Two-image matches (b) provides a larger number of matches but many outliers among them. Three-image matches (c) reduce the number of outliers considerably. For better visibility here 100 features are randomly selected and connected with the location of their matched features by a yellow line if such a feature is found.

feature. As features are only matched if they are likely correspondences in three images, the three matching algorithm obviously decreases the number of matches as compared to the algorithm that matches features based on two images. But our algorithm renounces to match many inconsistent features so that the percentage of outliers is greatly decreased. As we will show in the subsequent sections, this reduction of the relative amount of outliers allows matching based algorithms to start off much better.

5 APPLICATION TO STEREO-VIDEO CONSISTENT OPTICAL FLOW

Recent optical flow algorithms started to include feature matches into the dense correspondence estimation to faithfully detect large motion also of small objects. More specifically, Xu et al. consider motion vectors of matched features to possibly assign them to pixels all over the image [XJM10], whereas Brox et al. [BBM09] include matched regions as prior into their optical flow algorithm. We adopt the latter idea here and include matched features into the state-of-the-art optical flow for stereo sequences [SLM10]. This optical flow approach is derived from an optical flow algorithm [WTP+09] classified on the Middlebury benchmark [BSL+07]. It considers symmetry and consistency on a three image loop and therefore provides a suitable mean to evaluate the three image based matching. While in the approach of Brox et al. [WTP+09]



(a) (b) (c) (d)

Figure 4: For the scenes *art*, *laundry*, *waving*, *stonemill*, *Rubber Whale* and *Hydrangea* (a) dense ground-truth motion fields are given (b). Compared to the motion fields of the loop-consistent TV-L^2 algorithm of [SLM10], (c) the inclusion of our three-image match as prior results in motion fields with better motion detail (d).

several matches are considered to make sure that the correct correspondence is among them, we incorporate our matched features in their one-to-one fashion. Adopting the notation of $w^r_{i,j}$ for the current estimate of the motion field between image I_i and I_j we simply replace the point-wise energy E_q in [SLM10] with

$$E_f = E_q + \delta_f \|W_{i,j} - w^r_{i,j} - dw_{i,j}\|_2^2 \qquad (1)$$

where for matches $(f_{i,k}, f_{j,n}, f_{h,m})$ and $[f_{i,k}]$ the nearest integer position to the feature location

$$W_{i,j} : \Omega_i \to \mathbb{R}^2, \quad W_{i,j}(x) = \begin{cases} f_{j,n} - f_{i,k} & \text{if } x = [f_{i,k}] \\ 0 & \text{else} \end{cases} \qquad (2)$$

is a function that describes the matching of the features, $\mu, c > 0$ constants and

$$\delta_f(x) = \mu \begin{cases} 1 - \arctan \frac{\tilde{d}(s_{i,k}, s_{j,n})}{c2\pi} & \text{if } x = [f_{i,k}] \\ 0 & \text{else} \end{cases} \qquad (3)$$

a function that assigns values depending on the matching costs or 0 to each point in Ω_i. This new energy is still a quadratic function in the update $dw_{i,j}$, so the updating scheme of [SLM10] is maintained. Note that for all experiments we fix $\mu = 10^3$ and $c = \frac{1}{5}$

To speed up calculations and assist the determination of large flows, loop consistent flow estimation is performed on a factor 0.5 image pyramid. Similar to [BBM09] we down-sample the prior $W_{i,j}$ by considering the 2×2 pixels that are represented by one single pixel in the next coarser level. From the four pixels in the finer level we only pass on to the next coarser level half the motion and the weight of the pixel with the highest weight $\delta_f(x)$. Thus, if no other matches are found in the vicinity, the original match is propagated to the next coarser level or else the match with the smallest cost is used. Having thus established a matching-based prior on all levels of a scale pyramid, we initialize the dense flows on the coarsest level with zero and perform 10 iterations of the updating scheme before proceeding to the next finer level. We use the upscaled flow field from the previous level as initialization on the finer level and thus proceed till the original resolution is reached.

5.1 Evaluation

To evaluate the impact of three image-consistent matching on optical flow estimation, we use all the data sets with known ground-truth motion from Sect. 4 except for the scenes *graffiti* and *wall* which only contain camera motion around a planar scene and are therefore of no interest for dense motion field estimation. We measure the average angular error (AAE) and average endpoint error (AEE) [BSL$^+$07] between the computed and the ground-truth displacement fields. For comparison, we also calculate flow fields with a two-image TV-L^2 approach [SLM10] incorporating standard two image-feature matching as prior and the three image-loop consistent optical flow algorithm [SLM10] without prior. As SURF features provide the best cover of our test scenes with feature points, we here only show the results obtained with SURF. Flow fields incorporating priors obtained with other descriptors behave qualitatively in the same way:

If only two image matches and forward flow are considered, wrong matches have a strong impact and lead to results with high error, Tab. 2. In [SLM10] Sellent et al. show that loop consistent flow improves the results of the TV-L^2 approach. Incorporating feature points that are likewise consistent on three images is able to further improve the results. An improvement is also visible in the flow field, Fig. 4, as small structures such as e.g. the hand in the *waving* scene are better preserved than without the prior matches.

6 CONCLUSIONS AND FUTURE WORK

In our article we show that even in the absence of camera calibration and synchronization, feature points can be matched more robustly if three images are considered simultaneously. By requiring that features are consistent in three images, the quality of the matching improves as the percentage of wrong matches is considerably reduced.

We also combine three-image matching with three image-loop consistent optical flow estimation and obtain dense flow fields that have a smaller error and better preserved motion details than either the loop-consistent flow or basic flow with non-robustly matched features.

In this work we extend the traditional two image approach to three images and obtain more robust results. Future work in this direction compromises to evaluate whether this trend can be continued if four or more images are used and whether there is an optimal number of images to be used.

ACKNOWLEDGEMENTS

This work has been funded by the German Science Foundation, DFG MA2555/4-2.

REFERENCES

[ART10] A. Albarelli, E. Rodolà, and A. Torsello. Robust game-theoretic inlier selection for bundle adjustment. In *Proc. of the International Symposium on 3D Data Processing, Visualization and Transmission*, pages 1–8, Paris, France, May 2010.

[BBM09] T. Brox, C. Bregler, and J. Malik. Large displacement optical flow. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 41–48. IEEE, 2009.

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proc. of the European Conference of Computer Vision (ECCV)*, pages 25–36, 2004.

[BCS94] H. Bandelt, Y. Crama, and F. Spieksma. Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49(1-3):25–50, 1994.

[BETV08] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[BGPS07] S. Battiato, G. Gallo, G. Puglisi, and S. Scellato. SIFT features tracking for video stabilization. In *Proc. of the International Conference on Image Analysis and Processing*, pages 825–830, 2007.

[BSL$^+$07] S. Baker, D. Scharstein, JP Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Proc. ICCV*, pages 1–8. IEEE, 2007.

[BTZ96] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. In *Proc. of the ECCV*, volume 2, pages 683–695. Springer, 1996.

[BWSS09] X. Bai, J. Wang, D. Simons, and Guillermo Sapiro. Video snapcut: robust video object cutout using localized classifiers. *ACM Trans. Graph.*, 28(3):1–11, 2009.

	TV-L^2		TV-L^2 & 2IM		[SLM10]		[SLM10] & 3IM	
	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
art	1.68	10.62	49.45	84.82	1.32	9.34	1.09	8.70
books	11.23	14.60	31.99	55.37	2.67	6.43	1.62	4.85
dolls	1.93	5.81	32.86	67.66	0.53	2.85	0.51	2.27
laundry	7.83	14.16	40.38	58.08	1.27	9.20	1.03	8.39
moebius	0.96	3.67	16.85	23.77	0.87	3.61	0.87	3.13
reindeer	18.89	25.91	18.70	30.50	2.22	16.35	1.02	8.55
waving	2.95	1.03	26.96	31.39	2.74	0.97	2.58	0.92
stonemill	16.72	4.53	48.59	23.16	11.29	3.81	9.73	3.52
RubberW	6.60	0.20	15.07	5.72	6.46	0.20	6.34	0.20
Hydr.	2.98	0.27	9.28	4.39	2.79	0.25	2.77	0.24

Table 2: Including 2-image SURF matching priors (2IM) into TV-L^2 flow significantly increases average angular (AAE) and average endpoint error (AEE) in comparison to the basic TV-L^2 approach. Under consideration of consistency on a loop of three images, inclusion of 3-image SURF matching priors (3IM) decreases the AAE and AEE of the loop consistent TV-L^2 approach [SLM10].

[CS92] Y. Crama and F. C. R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60(3):273–279, 1992.

[FTV03] V. Ferrari, T. Tuytelaars, and L. Van Gool. Wide-baseline multiple-view correspondences. In *Proc. of the CVRP*, volume 1, pages 718–725, June 2003.

[HP96] A.Y.K. Ho and T.C. Pong. Cooperative fusion of stereo and motion. *Pattern Recognition*, 29(1):121–130, 1996.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the Alvey Vision Conference*, volume 15, pages 147–151, 1988.

[HZ03] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[LLM10] C. Linz, C. Lipski, and M. Magnor. Multi-image interpolation based on graph-cuts and symmetric optic flow. In *Proc. of the International Workshop on Vision, Modeling and Visualization*, page to appear, Siegen, Germany, November 2010. Eurographics, Eurographics.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2(60):91–110, 2004.

[MS05] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE T-PAMI*, 27(10):1615–1630, 2005.

[MTS+05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L.V. Gool. A comparison of affine region detectors. *Intern. Journal of Computer Vision*, 65(1):43–72, 2005.

[PS98] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Publications, Mineola, New York, USA, 1998.

[RD06] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV*, pages 430–443. Springer, May 2006.

[SLM10] A. Sellent, C. Linz, and M. Magnor. Consistent optical flow for stereo video. In *Proc. ICIP*, Sept. 2010.

[SLW+10] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor. Perception-motivated interpolation of image sequences. *ACM Transactions on Applied Perception*, pages 1–28, 2010.

[SP07] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proc. of the CVPR*, pages 1–8. IEEE Computer Society, June 2007.

[Spi00] F.C.R. Spieksma. Multi index assignment problems: complexity, approximation, applications. *Nonlinear Assignment Problems, Algorithms and Applications*, pages 1–12, 2000.

[SS05] K. Shafique and M. Shah. A noniterative greedy algorithm for multiframe point correspondence. *IEEE T-PAMI*, pages 51–65, 2005.

[SSS06] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics*, 25:835–846, July 2006.

[SZ02] F. Schaffalitzky and A. Zisserman. Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?". In *Proc. of the ECCV*, volume 1, pages 414–431. Springer, May 2002.

[TZ97] P.H.S. Torr and A. Zisserman. Robust parameterization and computation of the trifocal tensor. *Image and Vision Computing*, 15(8):591–605, 1997.

[VRB03] C.J. Veenman, MJT Reinders, and E. Backer. Establishing motion correspondence using extended temporal scope. *Artificial Intelligence*, 145(1-2):227–243, 2003.

[WTP+09] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *Proc. BMVC*, London, UK, Sept. 2009.

[XJM10] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. In *CVPR*, San Francisco, 2010. IEEE Computer Society.

[YC07] J. Yao and W.K. Cham. Robust multi-view feature matching from multiple unordered views. *Pattern Recognition*, 40(11):3081–3099, 2007.

[YJS06] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Computing Surveys*, 38(4):13, 2006.

[Zha94] Z. Zhang. Token tracking in a cluttered scene. *Image and Vision Computing*, 12(2):110–120, 1994.

[ZKP10] C. Zach, M. Klopschitz, and M. Pollefeys. Disambiguating visual relations using loop constraints. In *Proc. of the CVPR*, pages 1–9. IEEE, June 2010.

Accurate Real-Time Multi-Camera Stereo-Matching on the GPU for 3D Reconstruction

Klaus Denker
HTWG Konstanz, Germany
kdenker@htwg-konstanz.de

Georg Umlauf
HTWG Konstanz, Germany
umlauf@htwg-konstanz.de

ABSTRACT

Using multi-camera matching techniques for 3d reconstruction there is usually the trade-off between the quality of the computed depth map and the speed of the computations. Whereas high quality matching methods take several seconds to several minutes to compute a depth map for one set of images, real-time methods achieve only low quality results. In this paper we present a multi-camera matching method that runs in real-time and yields high resolution depth maps.
Our method is based on a novel multi-level combination of normalized cross correlation, deformed matching windows based on the multi-level depth map information, and sub-pixel precise disparity maps. The whole process is implemented completely on the GPU. With this approach we can process four 0.7 megapixel images in 129 milliseconds to a full resolution 3d depth map. Our technique is tailored for the recognition of non-technical shapes, because our target application is face recognition.

Keywords
Stereo-matching, multi-camera, real-time, gpu, computer vision.

1 INTRODUCTION

Stereo matching is a technique to compute depth information of a captured object or environment from two or more 2d camera images. Many applications ranging from remote sensing to robotics, archeology, cultural heritage, reverse engineering, and 3d face recognition [15, 17, 10, 26] use stereo matching. It is the only passive method to generate depth information. This means there is no artificial interaction with the object that might do any harm and only natural light is used for the data acquisition.

The main challenge of stereo matching is the trade-off between the quality of the depth map and the computation time to compute the depth map. For some applications a real-time computation is not important. So many stereo- and multi-view-matching methods focus on high quality results instead of fast computation times. These high quality methods need at least several seconds to compute a single depth map from one set of images [9]. However, for robotics faster computation times are more important than the quality of the depth map. This led to the development of GPU based real-time matching methods [28, 27].

Our target application is 3d face recognition. For face recognition the requirements are somewhere between these fields. A trade-off between a high depth map quality and an acceptable speed must be found. The whole reconstruction and recognition needs to be done in less than half a second. A longer delay is not acceptable for the captured person. Nevertheless, the quality of the reconstructed surface needs to be high enough for a reliable recognition of the person.

1.1 Overview and contribution

In order to classify our approach for the subsequent related work section we give here a brief layout of our system. It is based on weighted normalized cross-correlation for all matching windows of a reference image to a set of additional images from different perspectives. This cross-correlation yields a score for every matching window position and the maximal score indicates the best matching position. This best matching position corresponds to a disparity of the matching windows and thus to the depth information. These steps will be described in Sections 3 - 4. Our contribution in this process is the GPU optimized use of weighted normalized cross-correlations, the combination of multiple cameras to a total score for simultaneously moved matching window, a projection-free depth-map-based deformation of the matching windows, and a sub-pixel precise disparity estimation. These techniques account for the quality of the generated depth maps. To compute the depth maps in real-time our process is implemented on the GPU. This is described in Section 5 and has not be done in such a consequent form before.

2 RELATED WORK

Our method may be classified between two very different classes of stereo matching methods. On the one hand, the high quality methods with long computation time to achieve excellent results. On the other hand,

the fast GPU methods using much simpler algorithms. Therefore, we will contrast our approach to both classes of stereo matching methods.

2.1 High quality methods

High quality stereo matching methods have been developed based on various techniques. The quality of such methods is compared at [19, 21, 25]. Newer benchmark results are available on the associated websites [20, 22, 24].

One of the earliest methods in this class is the adaptive least squares correlation of [6]. In this approach local affine transformations are estimated using a least squares approximation. Although, this method theoretically converges to an optimal solution, the convergence is too slow and the computation too costly due to the size of the linear systems.

Today, best reconstruction quality is achieved by region growing algorithms, e.g. [5, 9]. These methods are typical for high quality matching algorithms, where a set of good matches is generated using a sparse set of interesting features. Then, these good matches are extended with a growing strategy. The growing operations are iterated in combination with filter operations to control the quality of the matches. Because the growing process is based on an optimization of complex objective functions, these methods do not allow a fast GPU implementation.

A novel alternative is the phase only correlation of [23]. Here, the disparity of matching windows is estimated by the phase difference of the image signal along epipolar lines. This requires the computation of a Fourier transformation, which is difficult to implement on the GPU [14]. This is particularly problematic if the Fourier transform must be evaluated for every pixel of the captured image.

Global optimization of a Markov Random Field (MRF) is used in [1]. For each pixel multiple depth hypotheses are stored and the best is picked by the MRF optimization. The solution of this NP-hard problem is approximated using a sequential tree re-weighted message passing algorithm [11]. Although the GPU is used to solve several steps of the algorithm, the global optimization makes it much slower than typical GPU methods.

A particle cloud optimization is used by [8] to generate depth representations for each camera image. The particles are aware of depth discontinuous silhouettes and use a special volumetric view space parametrization instead of the usual image-based parametrization of matching windows. Then, these depth representations are combined and rendered in real-time using the GPU.

Approaches based on dynamic programming, e.g. [12, 18], are relatively similar to our approach. For these methods fields of matching scores are computed for every epipolar line. Within these fields an optimal path is computed using dynamic programming. The computations of the optimal path can either be done on the CPU or on the GPU requiring significant amount of memory.

Our approach is also based on matching scores along epipolar lines, but the computations are local and simple to allow an implementation on the GPU.

2.2 GPU methods

Much faster methods implement the stereo-matching algorithm on the GPU using hardware features of the graphics card like mip-mapping.

A typical example for this class of methods is described in [27]. This approach consists of a set of individual steps of the overall stereo-matching process implemented on the GPU. For the matching score either the sum of squared differences or the sum of absolute differences are used. These matching scores are easily implemented on the GPU, but yield only low quality disparity maps. To exploit the capabilities of the mip-map a pyramidal matching kernel is used, which does not allow for an independent movement of the individual levels in the pyramid. In both aspects our approach improves this method. Some other aspects of [27], like cross-checking and feature aligned matching windows, could easily be integrated to our system.

A different approach of the same first author is [28]. Here five calibrated cameras are matched at once. Using the same technique with a reconfigurable array of 48 cameras is described in [30]. For this technique the matching window covers only one pixel to simplify the computations on the GPUs. This local approach is not stable but very fast and avoids all disadvantages of large matching windows.

Another technique for a large number of images is [29]. It is not as fast as the other GPU methods, but includes a volumetric reconstruction of the objects. A plane sweep method is used for depth estimation on non-rectified images.

The method from [2] uses the pyramidal matching kernel and mip-mapping from [27] and adds a foreground/background separation on the GPU. This additional step avoids typical artifacts of the pyramidal kernel like wrong depth estimates for regions with low texture details usually found in the background. Our improved multi-level approach does not show such problems.

3 THE CAMERA SYSTEMS

We built a system of four USB Logitech® QuickCam® Pro 9000 cameras, see Figure 1(a). Each camera is used at a resolution of 960×720 at five frames per second. The cameras could yield a much higher resolution, but the bandwidth of the USB 2.0 controllers is limited.

To improve the quality for later face recognition, we built a second camera system of four Point Grey Flea®2 FireWire 800 cameras, see Figure 1(b). These cameras synchronously capture images at a resolution of 1392×1032 at 15 fps. For synchronization we use all four cameras on a single FireWire 800 Bus. Thus, in RGB mode only a frame rate of 3.75 fps is possible. This can be improved by de-mosaicking on the GPU and transferring the data in eight bit raw mode. This allows for 11.25 fps.

Our experiments showed that a Y-constellation of four cameras as shown in Figure 1 gives the best results. The image of the central camera is used as reference image for matching and texturing. Each possible image pair has a different angle. Otherwise preferred directions of the camera constellation could deteriorate the detection of features along these directions, e.g. an image containing horizontal stripes causes problems for horizontal camera arrangements.

Independently of the used hardware system, our method can be adapted to other camera constellations. This adaption is much easier for camera systems where all cameras are mounted on a plane perpendicular to the viewing direction. The individual camera images are rectified using a lens correction similar to [4].

4 MATCHING

The overall matching process consists of several nested loops shown in Figure 3. We describe this process from the inner to the outer loop.

4.1 Stereo matching

The aim of stereo matching is to find corresponding points in two images. Usually two square regions, called *matching windows* are compared. These windows are moved over the images to find the best matching position. To identify the best position, a score is computed, that rates the similarity of two matching windows. Similar to [13] we use a *weighted normalized cross-correlation* on RGB color values. First the weighted average color $\overline{f}_i$ of the matching window W_i in the i-th image is computed

$$\overline{f}_i = \sum_{(x,y)\in W_i} w(x,y) f(x,y). \quad (1)$$

Here $w(x,y) = \cos^2(\pi x/a) \cdot \cos^2(\pi y/a)$ is a weight function that smooths the result to emphasize pixels at the center of the matching window over pixels at the border, and a denotes the matching window size in pixels. Then the weighted auto-correlation α_i of each matching window with itself is computed as

$$\alpha_i = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right]^2. \quad (2)$$

To evaluate the similarity of two matching windows W_i and W_j the weighted cross-correlation $\beta_{i,j}$ is computed

$$\beta_{i,j} = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right] \cdot \left[f_j(x,y) - \overline{f}_j \right]. \quad (3)$$

The weighted normalized score $\gamma_{i,j}$ is computed as the weighted cross-correlation normalized by the geometric mean of the respective weighted auto-correlations

$$\gamma_{i,j} = \beta_{i,j} / \sqrt{\alpha_i \cdot \alpha_j}. \quad (4)$$

4.2 Multi-camera matching

Stereo matching evaluates the similarity of two matching windows. We extend this score to a set of n cameras and matching windows by summing up the weighted normalized scores of all possible image pairs. Thus, we need $n(n-1)/2$ stereo matching operations. To compute a total score we compute a camera score

$$\gamma_i = \sum_{j \neq i} \gamma_{ij} \quad (5)$$

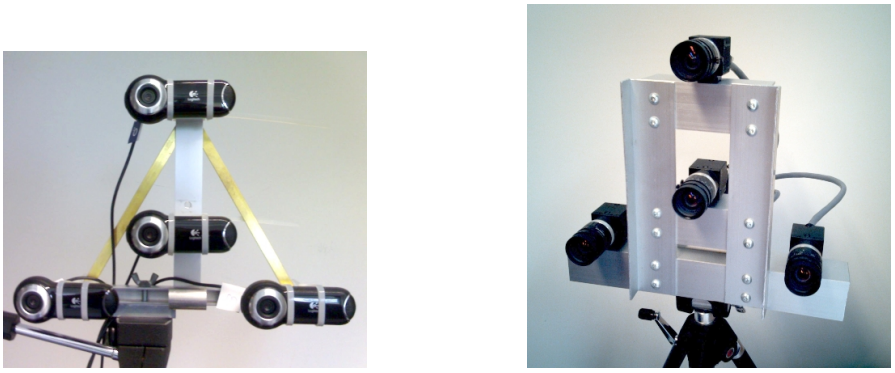
and a total score

$$\gamma = \sum_i \gamma_i - 2 \min_i \gamma_i. \quad (6)$$

This eliminates all scores from the worst matching camera to improve robustness to occlusion on one of the cameras. The total score is used to evaluate the similarity of matching windows of multiple cameras simultaneously.

4.3 Moving the matching windows

Between the images a disparity estimation is computed to get the depth information. Therefore, the matching windows are moved simultaneously over all images. A total score of each position and the best matching window position with the highest total score are computed. Since the evaluation of all possible positions is too expensive, the movement of the matching window is limited to the epipolar lines projected by the center point of the matching window of the reference image. The image of the central camera is used as reference image, i.e. the matching window on the central image is fixed. Figure 2 shows the simultaneous movement of the matching windows in the other images along the epipolar lines. These movements along the epipolar line have a step size of one pixel for our camera configuration. For other camera configurations the step size depends linearly on the distance to the central camera. We test $3 \leq k \leq 35$ different positions for each matching window, see Section 4.5. Note that the color values for the score computations are bi-linearly interpolated to allow an exact movement along the epipolar line. The best similarity of the matching windows is marked by the matching window position with the highest score



(a) USB camera system. (b) FireWire camera system.

Figure 1: For our experiments we use two systems of four cameras arranged in an upside down Y-constellation.

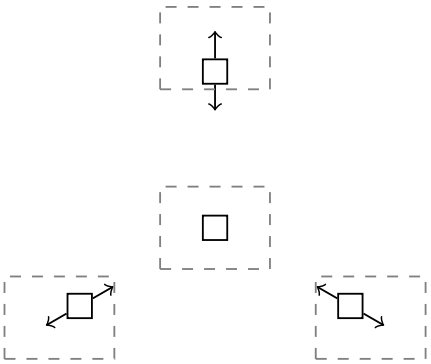


Figure 2: Moving the matching windows (solid squares) in all images (dashed rectangles) along epipolar lines (arrows) simultaneously.

s_{best}. From the position on the epipolar line, the disparity d_{best} of the best match is estimated. The real depth can be computed by reverse projection using the position of the reference camera, the distances to the other cameras, and the disparity.

4.4 Sub-pixel matching

To achieve sub-pixel precision for the disparity map we use a method similar to sub-pixel accurate edge detection of [3]. The best disparity is achieved at a local maximum of the total score, i.e. both neighboring scores s_{left} and s_{right} are smaller or at most one of them is equal to s_{best}

$$s_{\text{left}} \leq s_{\text{best}} > s_{\text{right}} \quad \text{or} \quad s_{\text{left}} < s_{\text{best}} \geq s_{\text{right}}. \qquad (7)$$

Interpolating these three total scores with a quadratic polynomial yields a best sub-pixel score at the global maximum of the quadratic polynomial. This maximum is achieved within half the distance to the neighbor positions. The position of this maximum is the interpolated sub-pixel disparity d_{sub}.

4.5 Multi-level matching

Our method generates disparity data for one image at a fixed resolution. To allow large disparities, many possible matching window positions must be evaluated. Because this is computationally expensive, we use a real multi-level approach that can reduce the effort for large disparities. A similar approach in [27] uses a matching pyramid. In contrast to our method, the windows on different detail levels cannot be moved independently.

Independent levels allow us to re-use high level information to get a much faster low level disparity computation. The graphics card stores the lens corrected image in a mip-map at eight different resolutions. Each level has half the horizontal and vertical resolution of the one below. All matching windows have a fixed size of 7×7 pixels. A smaller window size increases the noise while a larger size blurs sharp features. Starting on the coarsest resolution level $l = 7$, the disparities of all pixels in the reference images are computed at the same coarse resolution. The matching windows are evaluated at $k = 35$ different positions. Then the image resolution is doubled and the same process starts again, while $k = 1 + 2\lfloor 1.5 + l^2/3 \rfloor$ is reduced quadratically. As starting position for the matching windows on lower levels, the bi-linearly interpolated disparities of the next coarser level are used. Thus, the matching window moves k pixels around the best position of the previous level.

4.6 Deformed matching windows

Square matching windows can only yield good results, if the captured object surface is parallel to the image plane. Every surface not parallel to the image plane generates imprecision. To avoid this the matching windows are deformed to fit the perspective deformation of the object surface. The idea is based on [7], but we use the multi-level depth information and a projection free computation.

To estimate the deformation we use the disparity map of the previous multi-level step. First nine disparity values at the corners, the edge midpoints, and the center of the matching window are interpolated. This gives a disparity estimate for every pixel in the actual matching

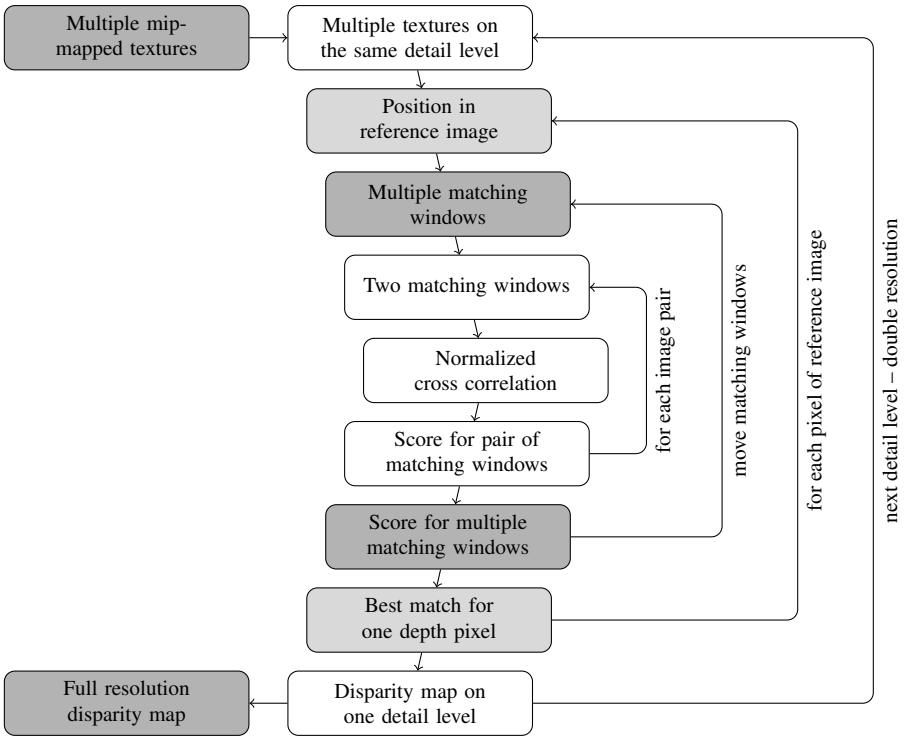


Figure 3: Overview of our matching process.

window. Subtracting the disparity at the center of the matching window yields a local displacement for every pixel. This displacement is added to the pixel coordinates before the color values are read. This results in a matching window adapted to the perspective of the previous level without computing any perspective projections. Note, that for planar object surfaces this approach is almost equivalent to the projections used by [7]. The difference is that it is based on disparity instead of depth.

4.7 Measuring the matching quality

For each resolution level a complete disparity map is computed. So, for each pixel of this map the best total score computed is also stored. Averaging these total scores over multiple resolution levels gives a quality measure for each pixel of the full resolution depth map, see Figure 4(b). Pixels with low quality measures can be masked for rendering or subsequent computations of the user application.

The quality measure is also used to improve the performance of the multi-level matching. A low quality measure on a coarse matching step usually causes the finer level matches in this region to fail too. Matching calculations are skipped if the quality measure on the next coarser level is too low.

5 IMPLEMENTATION ON THE GPU

The method described so far uses images and generates a depth image as result. Therefore, we use GLSL fragment shaders for the GPU implementation. A fragment shader is a program that runs in parallel on the GPU and processes one or multiple texture images into one result image. For our shader operations we need GPUs which support at least shader model 4.0. The required amount of computations in a single shader run is not feasible on older GPUs.

5.1 GPU lens correction

Our input data are multiple raw camera images. Each raw image is corrected by a shader implementing a lens correction similar to [4]. The resulting corrected images are rendered into separate textures. Each of these textures is then transformed into a mip-map. These mip-maps of the corrected images are used by all subsequent shaders of our system.

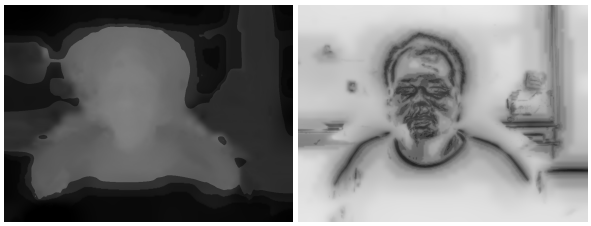
5.2 GPU optimized matching

A single pixel shader run usually computes the color values for one result image, each pixel separately. More complex computations require the combination of multiple shader runs. Three fragment shader programs are used for each step of our multi-level matching.

The first shader takes the corrected image mip-map and computes the weighted average color of the pixels of a matching window at the actual resolution level. These averages are rendered to separate average textures. This shader is invoked once for every image.

The second shader takes the corrected image mip-map and the average texture and computes the weighted auto-correlation for the same matching window. Again

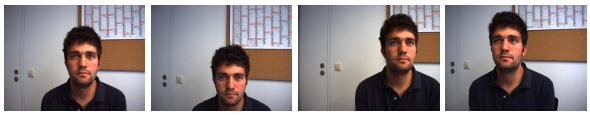
(a) The four captured sample images.



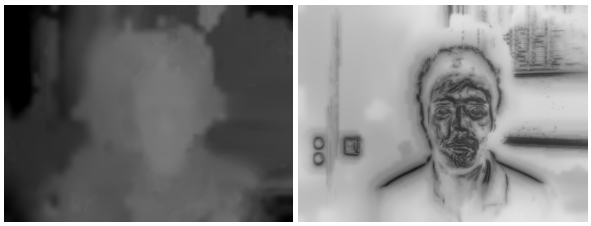
(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 4: Example from our USB camera system.



(a) The four captured sample images.



(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 5: Example from our FireWire camera system.

the result is rendered to a separate auto-correlation texture and the shader is invoked once for every image.

The third shader takes the average and auto-correlation textures and performs all matching operations, i.e. it moves the deformed matching windows, computes the total score, and finds the best sub-pixel score. The result is rendered as the disparity map, the best total score of the finest resolution and the quality measure to the three color channels of a separate texture. These three shaders are invoked once per resolution level.

Most important strategies used to improve the GPU performance are the pre-calculation of weighted average and weighted auto-correlation just described and the multi-level matching described in Section 4.5.

6 RESULTS

Our target application is face recognition. We present our results in that area. For easier comparison with other algorithms we also applied our algorithm to a well known benchmark for stereo matching.

6.1 Face reconstruction

We took some example images with our USB camera system shown in Figure 4(a). The disparities between these images are very large. The result texture of the fragment shaders holds the disparity map, the best total score of the finest resolution level, and the quality measure encoded in the color channels, see Figure 4(b).

After transformation of the disparities to depth values, the data can be rendered as 3d model, see Figure 4(c). The low quality regions are masked and ignored in this rendering.

A typical problem of stereo matching can be seen at the highlights on the forehead generating small dents, because the reflection is further away from the cameras than the forehead. More diffuse lighting could avoid this problem. The computation for the example images takes an average processing time of 129 ms on an NVidia GeForce GTX 285 GPU. This allows real-time frame rates of 7.5 fps.

A higher resolution of 1392×1032 is achieved by the FireWire camera system. An example image set is shown in Figure 5(a). Figure 5(b) shows the result textures and Figure 5(c) a 3d model of the resulting depth

map. The higher camera resolution yields a better shape quality at the most important regions of the face. Especially the reconstruction of the eye and mouth regions is much more precise.

For this example an average processing time of 263 ms is needed on the same GPU. For images of 30 different persons we get an average processing time of 254 ms. In most of these images the face region is smaller than in the displayed examples, so the computations are a bit faster. In comparison to the first example, the computation time grows almost linearly with the number of pixels p. This conforms to a runtime of $O(p \log p)$ for our multi-level algorithm: The matching window size, the stretch of the window movement, and the count of image pairs are constant. So the worst case costs for the computations in each depth map pixel is constant. The pixels of the resulting depth map, or smaller versions of it, are computed once for each of the $\log_2(\text{width}) \in O(\log p)$ multi-level steps. Hence the overall count of pixel calculations and the complexity of the algorithm is within $O(p \log p)$.

6.2 Stereo vision benchmarks

Several benchmarks can be used to compare the quality of stereo matching algorithms [19, 21, 25]. Our algorithm is tailored to face reconstruction and contains simplifications that require a planar camera configuration. Thus, it is not comparable to the benchmark [25]. Furthermore, our algorithm is also tailored to large disparities between the images and achieves a much better reconstruction quality using more than two cameras. So, only a comparison with the results of the extended datasets of the Middlebury stereo benchmark [20] is relatively fair. However, this benchmark does not provide an official score.

Compared to the algorithms providing results and timings for these benchmark our algorithm works much faster. At the same time the quality of our result is comparable to the quality of these algorithms. However, for this comparison we have to adapt our algorithm.

For the Middlebury stereo evaluation [20], we integrated a modified local version of Multi Hypothesis Matching [1] to improve the sharpness of edges in our algorithm. The movement range of the matching windows is extended to the depth extrema of the local neighborhood on the last detail level. Instead of evaluating only the best matching score, the eight best matching scores are stored. A post-processing step re-weights these scores based on the values and depth distances to the best scores in the direct pixel neighborhood. The re-weighting is repeated two times without any global optimization as in [1]. This multi hypothesis matching is implemented as an post-processing fragment shader on the GPU. The additional shader and the increased range for the matching windows cause a large performance loss. Processing our example images at a resolution of 960×720 pixels takes 900 ms. This is still faster than the other algorithms in [20], but not fast enough for our target application.

Figure 6 shows our algorithm applied to the extended *Tsukuba* dataset from [20, 16]. The two images in Figure 6(b) show the results from all five input images without and with the additional edge improvement.

7 CONCLUSION AND FUTURE WORK

The quality of the resulting surface model is sufficient and the processing times are more than sufficient for our target application 3d face recognition. Additional methods like cross-checking that can be implemented on the GPU could further improve our results. Furthermore, for an application of our method in a face recognition system, a simple method to guide persons to the optimal distance from the camera system is required.

For the future we plan to record synchronous video sequences with the FireWire camera system. Similar to multi-level matching, the matching information of an earlier video frame could be used to improve the performance.

ACKNOWLEDGMENTS

This work was supported by DFG GK 1131 and AiF ZIM Project KF 2372101SS9. We thank Jens Hensler for his help on creating a collection of face pictures.

REFERENCES

[1] Neill D. Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Proc. of the 10th European Conf. on Computer Vision*, pages 766–779, 2008.

[2] Jia-Ching Cheng and Shin-Jang Feng. A real-time multiresolutional stereo matching algorithm. In *ICIP (3)*, pages 373–376, 2005.

[3] F. Devernay. A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical Report RR-2724, INRIA, 1995.

[4] F. Devernay and O. Faugeras. Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl.*, 13(1):14–24, 2001.

[5] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. In *CVPR*, pages 1–8, 2007.

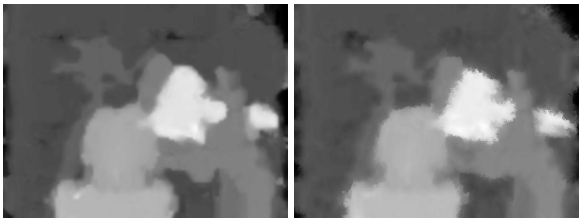
[6] A W Gruen. Adaptive least squares correlation: A powerful image matching technique. *South African Journal of Photogrammetry, Remote Sensing and Cartography*, 14:175–187, 1985.

[7] Hiroshi Hattori and Atsuto Maki. Stereo matching with direct surface orientation recovery. In *In Ninth British Machine Vision Conference*, pages 356–366, 1998.

[8] Alexander Hornung and Leif Kobbelt. Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling. *Comp. Graph. Forum*, 28(8):2090–2103, 2009.

[9] Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proc. of the 18th Int. Conf. on Pattern Recognition*, pages 15–18, 2006.

(a) The extended *Tsukuba* dataset pictures.



(b) Result disparity map of our algorithm without (left) and with edge enhancement (right).

(c) Ground truth disparity map (left) and 3d rendering of the result with edge enhancement (right).

Figure 6: Results of the extended *Tsukuba* dataset from the Middlebury stereo benchmark [20].

[10] Reinhard Koch, Marc Pollefeys, and Luc Van Gool. Realistic 3-d scene modeling from uncalibrated image sequences. In *ICIP'99, Kobe: Japan*, pages 500–504, 1999.

[11] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[12] Cheng Lei, Jason Selzer, and Yee-Hong Yang. Region-tree based stereo using dynamic programming optimization. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2378–2385, 2006.

[13] J. P. Lewis. Fast template matching. In *Vision Interface*, pages 120–123, 1995.

[14] Kenneth Moreland and Edward Angel. The FFT on a GPU. In *Proc. of the ACM Conf. on Graphics Hardware*, pages 112–119, 2003.

[15] Don Murray and Jim Little. Using real-time stereo vision for mobile robot navigation. In *Autonomous Robots*, pages 161–171, 2000.

[16] Yuichi Nakamura, Tomohiko Matsuura, Kiyohide Satoh, and Yuichi Ohta. Occlusion detectable stereo – occlusion patterns in camera matrix. In *CVPR*, pages 371–378, 1996.

[17] D.T. Pham and L.C. Hieu. Reverse engineering - hardware and software. In V. Raja and K.J. Fernandes, editors, *Reverse Engineering - An Industrial Perspective*, pages 33–30. Springer, 2008.

[18] H. Sadeghi, P. Moallem, and S. A. Monadjemi. Feature based dense stereo matching using dynamic programming and color. *International Journal of Computational Intelligence*, 4(3):179–186, 2008.

[19] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[20] D. Scharstein and R. Szeliski. Middlebury stereo vision page. `http://vision.middlebury.edu/stereo/`, 2007.

[21] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 519–528, 2006.

[22] Steve Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. Multi-view stereo. `http://vision.middlebury.edu/mview/`, 2009.

[23] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi. A sub-pixel stereo correspondence technique based on 1d phase-only correlation. In *ICIP07*, pages 221–224, 2007.

[24] Christoph Strecha. Multi-view stereo test images. `http://cvlab.epfl.ch/~strecha/multiview/`, 2008.

[25] Christoph Strecha, Wolfgang von Hansen, Luc J. Van Gool, Pascal Fua, and Ulrich Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*, pages 1–8. IEEE Computer Society, 2008.

[26] Francesca Voltolini, Sabry El-Hakim, Fabio Remondino, and Lorenzo Gonzo. Effective high resolution 3d geometric reconstruction of heritage and archaeological sites from images. In *Proc. of the 35th CAA Conference*, pages 43–50, 2007.

[27] Ruigang Yang and Marc Pollefeys. A versatile stereo implementation on commodity graphics hardware. *Real-Time Imaging*, 11(1):7–18, 2005.

[28] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proc. of the 10th Pacific Conf. on Computer Graphics and Applications*, pages 225–235, 2002.

[29] Christopher Zach, Mario Sormann, and Konrad F. Karner. High-performance multi-view reconstruction. In *3DPVT*, pages 113–120, 2006.

[30] Cha Zhang and Tsuhan Chen. A self-reconfigurable camera array. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 151, 2004.

Ambient Occlusion Opacity Mapping for Visualization of Internal Molecular Structure

David Borland
The Renaissance Computing
Institute, USA
borland@renci.org

ABSTRACT

Molecular surfaces often exhibit a complicated interior structure that is not fully visible from exterior viewpoints due to occlusion. In many cases this interior cavity is the most important feature of the surface. Applying standard blended transparency can reveal some of the hidden structure, but often results in confusion due to impaired surface-shape perception. We present ambient occlusion opacity mapping (AOOM), a novel visualization technique developed to improve understanding of the interior of molecular structures. Ambient occlusion is a shading method used in computer graphics that approximates complex shadows from an ambient light source by rendering objects darker when surrounded by other objects. Although ambient occlusion has previously been applied in molecular visualization to better understand surface shape, we instead use ambient occlusion information to determine a variable opacity at each point on the molecular surface. In this manner, AOOM enables rendering interior structures more opaque than outer structures, displaying the inner surface of interest more effectively than with constant-opacity blending. Furthermore, AOOM works for cases not handled by previous cavity-extraction techniques. This work has been driven by collaborators studying enzyme-ligand interactions, in which the active site of the enzyme is typically formed as a cavity in the molecular surface. In this paper we describe the AOOM technique and extensions, using visualization of the active site of enzymes as the driving problem.

Keywords: Molecular visualization, ambient occlusion, transparent surfaces

1 INTRODUCTION

Visualization has become an essential tool for many scientists working with molecular data. Ball-and-stick, ribbon, and surface renderings are all visualization techniques that enable improved understanding of molecular structures [10, 23, 30, 40]. Our collaborators use visualization techniques such as these to study enzyme-ligand interactions.

Enzymes are proteins that catalyze the transformation of a substrate molecule into a product. The substrate/product is often referred to as the "ligand" with which the enzyme binds. Our collaborators use molecular surface renderings to understand the shape of the active site of the enzyme and its spatial relationship to the ligand during binding. The active site is typically a complicated internal cavity that is largely hidden from exterior viewpoints due to occlusion (Figure 2). Two tools commonly used for viewing occluded structures are transparency and clipping planes.

Applying standard blended transparency to the surface reveals some of the internal structure, but often results in impaired surface-shape perception [24]. Clipping planes can also reveal internal structure, but are typically insufficient for displaying complicated non-planar surfaces such as enzyme active sites. Another potential technique that might be used is applying depth-peeling to remove the closest surface to the viewpoint. Such a technique would be inaccurate for this application, however, as the visible portions of the cavity would be removed, and the second surface may not correspond to the cavity in areas of folds and bumps of the outer surface. To enable improved visualization of the active sites of enzymes, we have developed a technique that uses ambient occlusion information to identify and emphasize these hidden structures.

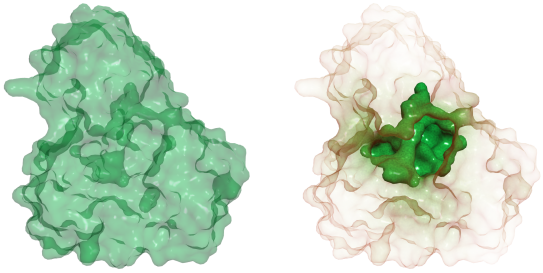


Figure 1: Standard transparency on the left versus ambient occlusion opacity mapping (AOOM) on the right. The structure of the inner cavity, and its context within the outer surface, is easier to understand with AOOM.

Ambient occlusion is a shading method used in computer graphics that approximates complex shadows from an ambient light source. Surfaces surrounded by objects that block ambient light are rendered

darker than those open to the environment, so ambient occlusion is therefore a measure of the "hiddenness" of an object. Ambient occlusion has been successfully used in molecular visualization to help understand surface shape and identify the locations of cavities in molecular surfaces [39]. Instead of using ambient occlusion information solely for enhancing surface shading, ambient occlusion opacity mapping (AOOM) uses ambient occlusion to calculate a variable opacity at each point on the molecular surface. Because ambient occlusion is a measure of the hiddenness of an object, AOOM can render outer structures more transparent and inner structures more opaque, displaying the inner surface of interest more effectively than with standard transparency (Figure 1). In this paper we apply AOOM to the visualization of enzyme active sites and describe various extensions to the core AOOM technique driven by collaboration with biochemists.

The paper is organized as follows: Section 2 provides background information on the biochemistry our collaborators are studying. Section 3 provides related work in the areas of molecular surface visualization, occlusion and transparency in visualization, and ambient occlusion. Section 4 describes the AOOM implementation, extensions, and supplemental visualization techniques. Section 5 concludes and provides future work.

2 SCIENTIFIC BACKGROUND

Our collaborators study the architecture of the active sites of enzymes involved in the tetrapyrrole biosynthesis pathway. The enzymes in this pathway catalyze reactions involving ligands that are necessary for the formation of various molecules such as hemoglobin, vitamin coenzyme B_{12}, and chlorophyll. Of interest are how different enzyme active-site architectures interact with their respective ligands.

PyMOL, an open-source molecular visualization system (www.pymol.org), is used by our collaborators to visualize ligands bound in the enzyme active sites, with crystallographic data taken from the Protein Data Bank (www.pdb.org). Understanding the active site where the ligand binds with the enzyme is especially important for answering questions such as: a) How much space is available for the ligand within the volume occupied by the protein? b) How does the ligand access the active site cavity? c) How completely is the active site cavity filled by the ligand? and d) Which residues in the cavity are close enough to the ligand to provide the anchoring interactions that bind it in place?

Each of these questions involves understanding the shape of the active site cavity, which can be problematic due to self-occlusion of the inner cavity by the outer surface (Figure 2). Dealing with occluded surfaces has long been an area of research in visualization. The next section provides previous work on molecular surface visualization, visualizing occluded surfaces, and ambient occlusion.

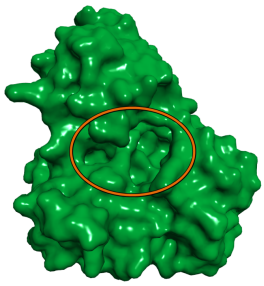


Figure 2: The surface cavity forming the active site of the PGB deaminase enzyme is circled. Much of the cavity is occluded by the outer surface.

3 PREVIOUS WORK

3.1 Molecular Surface Visualization

Molecular surfaces are a common visualization technique for studying molecular structures. The solvent-excluded molecular surface is formed by rolling a spherical probe over spheres representing the atoms of the molecule [34]. It represents areas accessible by molecules of a given probe radius. Connolly described methods for generating these surfaces [8, 9]. Methods improving the efficiency and quality of computing these surfaces have also been described [1, 6, 36, 41]. While such methods for producing molecular surfaces are necessary for the visualizations produced in this paper, they do not address the problem of visualizing the occluded interior structure of the generated surfaces.

The problem of visualizing protein docking using surfaces has been addressed [27]. This approach computes the intermolecular negative volume of two docked proteins to determine the amount of intersection between the two surfaces, with the purpose of enhancing drug-design by testing different potential conformations. While effective for such work, this approach is not directly applicable to the biochemistry presented in this paper, which involves data with known structure and no surface intersection.

Methods for analytically extracting pockets and cavities using computational geometry techniques also exist. CASTp uses the weighted Delaunay triangulation and alpha complex to identify and measure the area and volume of pockets and cavities [17], and is available as a PyMOL plugin. The ability to extract measurements of pockets and cavities is very useful, however comparison with an AOOM rendering demonstrates that important features of the cavity may be missed, such as the circled portion of the cavity on the left and the circled access tunnel on the right in Figure 3, Bottom Left (Bottom Right includes a Focal Region technique dis-

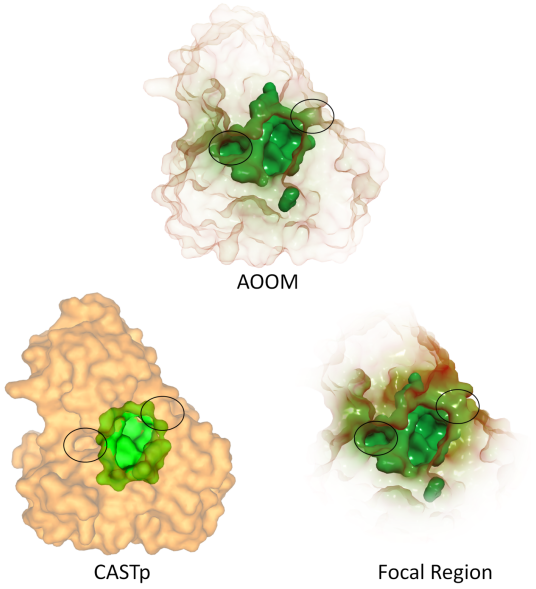


Figure 3: CASTp cavity extraction (bottom left, rendered using PyMOL) does not extract the entire cavity, and does not conform to the original molecular surface (missing circled regions). A focal-region approach (bottom right) based on distance from the center is less effective than AOOM (top), as it occludes portions of the inner cavity (circled access tunnel on the right) and erodes regions of the outer surface that otherwise provide visual context.

cussed in Section 3.2), because the cavity is not calculated from the full molecular surface, but instead from the extracted atoms that form the cavity. Also, there are a number of structures for which CASTp fails, whereas AOOM will work for any molecular surface (Figure 4). Future work will include augmenting AOOM with the types of analytical capabilities provided by CASTp. A promising step in this direction involves extracting the cavity by thresholding based on ambient occlusion information, followed by connected components analysis to remove smaller pockets in the surface (Figure 4, Bottom).

Recent work has resulted in techniques for producing simplified abstractions of complicated molecular surfaces [7]. While this technique does not directly address revealing hidden internal structure, it may prove beneficial to combine AOOM with such abstracted surfaces, as AOOM will work with any surface-based representation. Furthermore, AOOM could be applied to other representations, such as ball-and-stick and ribbon renderings.

3.2 Occlusion and Transparency

Occlusion is the most powerful of all depth cues [43]. However, occlusion can be problematic when visualizing 3D data, as objects of interest can be hidden from view. Applying transparency to occluding objects can

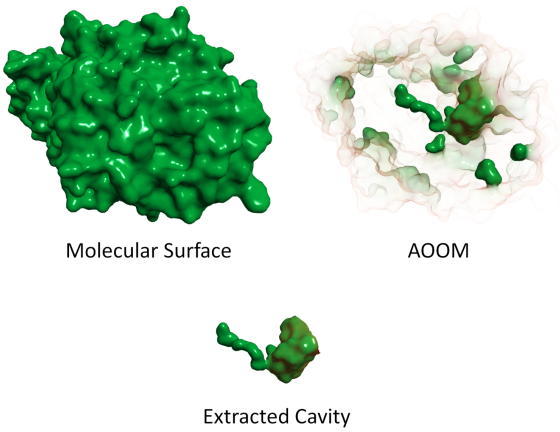


Figure 4: A molecular structure [35] (top left) for which CASTp fails, that reveals a long tube-like cavity structure when rendered using AOOM (top right). If desired, the cavity can be extracted by thresholding on ambient occlusion information followed by connected components analysis (bottom).

make hidden objects visible, but simple transparent surfaces do a poor job of conveying surface shape [24]. Various techniques have therefore been developed to enable more effective visualization of occluding and occluded surfaces.

Illustrative Techniques Illustrative techniques include a surface-rendering technique for view-dependent tranparency that aims to automatically produce transparent surfaces similar to technical illustrations [15]. Later work describes techniques for automatically producing breakaway and cutaway illustrations of nested surfaces [16]. These techniques are useful for nested surfaces, but do not address a single self-occluding surface. Similar work has been applied to isosurfaces extracted from volumetric data [19], which is useful for objects within the volume of the isosurface, but not for a single self-occluding surface.

Focus-and-Context Techniques A class of direct volume rendering techniques has been developed to reveal the inner structure of volumes while retaining some outer structure to maintain context. Importance-driven volume rendering highlights presegmented regions based on user-supplied importance criteria [42]. Opacity reduction of occluding volumes by finding volumetric features indicating a separation between areas with similar voxel intensities has been described [2, 3]. Selective opacity reduction of regions using a function of shading intensity, gradient magnitude, distance to the eye point, and previously accumulated opacity has also been described [5]. Opacity-peeling can be used to remove some fixed number of fully-opaque layers of material [32]. These focus-and-context techniques use various features of the volumetric data to modulate opacity and reveal hidden structure, and therefore

do not apply directly to molecular surface rendering. A depth-dependent focal region can also be used for opacity modulation [29]. However, applying a similar technique to our data shows that it is insufficient by itself due to the irregular geometries formed by molecular surfaces, even for a relatively round and centralized cavity (Figure 3, Bottom Right). The work of [11] is closest to that described in this paper, as ambient occlusion is also used for modulating opacity. However, their work focuses on volumetric data, and does not perform the smoothing process described in 4.2 (Figure 6 shows AOOM without smoothing).

3.3 Ambient Occlusion

For the techniques listed above, some method of determining the object or volume of interest is necessary, either via distinct and separate surfaces or via functions of the underlying volume. For displaying the inner cavity of a molecular surface, we need a means to determine which portions of the surface constitute the inner cavity of interest. To do so we calculate ambient occlusion for the surface.

Ambient occlusion [4, 26] is a shading method used in computer graphics that approximates complex shadows from an ambient light source by rendering objects darker when surrounded by other objects (Figure 5, Left). The basic algorithm involves casting a number of rays at various angles from each point on a surface, keeping track of the number of rays that intersect another (or the same) surface. Recent work has focused on real-time ambient-occlusion calculation for dynamic scenes [20, 25, 37, 38], but for our static surfaces it is sufficient to use a pre-calculated ray-tracing approach and store the ambient occlusion per-vertex. The ambient occlusion term O_p at a point p on a surface with normal N can be computed by integrating the visibility function V_p over the hemisphere Ω with respect to the projected solid angle:

$$O_p = \frac{1}{\pi} \int_{\Omega} V_p(\vec{\omega})(N \cdot \vec{\omega})\, d\omega, \tag{1}$$

where $V_p(\vec{\omega})$ is zero if p is occluded in the direction $\vec{\omega}$, and one otherwise. The dot product $N \cdot \vec{\omega}$ results in a cosine-weighting across the hemisphere. Using a cosine-weighted distribution of rays therefore removes the need for this cosine factor, resulting in a simple ratio of the number of rays that intersect a surface r_i and the number of total rays r_t:

$$O_p = \frac{r_i}{r_t}. \tag{2}$$

Areas of the surface that are largely occluded will therefore have a high O_p value.

Ambient occlusion rendering was developed to enhance realism in computer graphics by replacing the standard ambient term by $1 - O_p$ to darken objects blocked from ambient light. Ambient occlusion has also proven useful for scientific visualization. For example, with molecular surface rendering, the locations of cavities in the molecular surface become more apparent (Figure 5, Left). Because ambient occlusion is a measure of the "hiddenness" of a particular point on a surface, we can use ambient occlusion information to identify hidden structures and render them more opaquely to provide visual emphasis.

4 AMBIENT OCCLUSION OPACITY MAPPING

Ambient occlusion opacity mapping (AOOM) uses ambient occlusion information to modulate the opacity of the molecular surface. Areas with high ambient occlusion that would typically be rendered dark, are instead rendered more opaque than areas with low ambient occlusion. The color of the surface can also be adjusted based on ambient occlusion to enhance perception of the inner cavity versus the outer surface. This section describes the implemenation details of AOOM, extensions to the core AOOM technique enabling enhanced opacity control, and supplemental visualization techniques used with AOOM to visualize enzyme-ligand interactions.

4.1 Implementation Details

The examples shown here use surfaces exported from PyMOL. A molecular surface with a probe size of 1.4 Å ($\approx$ radius of water) is used. For most of the examples in this paper, we show PBG deaminase [28] for consistent comparison. Figures 4 and 12 show AOOM applied to other molecules.

We have implemented AOOM via extensions to the Visualization Toolkit (VTK) (www.vtk.org). Surfaces exported from PyMOL are loaded in OBJ or VRML format. Ambient occlusion is pre-computed on the CPU for each vertex on the input via a ray-casting approach, and stored as scalar point data. As with other ray-tracing techniques, calculating per-vertex ambient occlusion is embarrassingly parallel, so we accelerate computation by distributing computation across processing units. Results are typically stored in one of VTK's polygonal file formats so that computation is only necessary once. Because the ambient occlusion is pre-computed, there is a neglible decrease in performance when rendering with AOOM.

Depth sorting is performed to obtain correct blending, which can affect performance for large surfaces, however this is also the case for standard transparency. A depth-peeling approach could also be used to achieve order-independent transparency [14, 18]. Because the O_p term constitutes a scalar field mapped to the surface, a full range of color and opacity functions can be applied, typically in the same fragment program used for lighting and shading. In the simplest case, the opacity

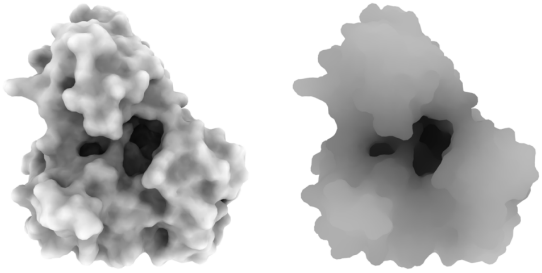


Figure 5: Ambient occlusion (left) vs. smoothed ambient occlusion (right). Using smoothed ambient occlusion for opacity enables filtering of small-scale concavities in the surface.

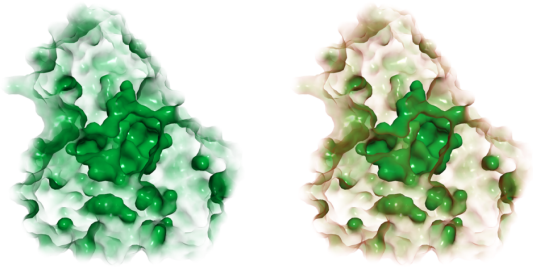


Figure 6: The simplest implementation of AOOM, in which ambient occlusion is directly mapped to opacity. The image on the left uses a constant color, and the image on the right applies a color map to the ambient occlusion.

is set equal to the O_p ambient occlusion term, $\alpha = O_p$, via a linear lookup table, and a constant color is used (Figure 6, Left). A double-ended color map that separates high and low ambient occlusion values can also be applied to help viewers distinguish interior and exterior structures (Figure 6, Right). For the examples in this paper we apply a color map from orange (low ambient occlusion) to green (high ambient occlusion).

This approach is more effective than standard transparency (Figure 1, Left) in revealing the structure of the inner cavity, but can be improved upon with additional opacity controls.

4.2 Smoothed Ambient Occlusion

Although Figure 6 demonstrates an improvement over standard transparency techniques, there are still areas of the outer surface that occlude the interior cavity, due to small concave pockets formed on the surface. To deemphasize these pockets, we smooth the ambient occlusion data over the surface. Smoothing the ambient occlusion filters out small-scale features, while retaining the larger cavity (Figure 5). We smooth the ambient occlusion data directly on the surface by iteratively solving the diffusion partial differential equation:

$$\frac{\partial u}{\partial t} = D\,\nabla^2 u, \tag{3}$$

where D is a constant controlling the amount of diffusion per time step ($= 1$ in the general case). This approach is equivalent to smoothing using a Gaussian filter, with more iterations equal to a Gaussian with a larger standard deviation. We solve the diffusion equation rather than performing direct convolution with a Gaussian because solving the diffusion equation iteratively only requires sampling immediate neighbors in the polygonal mesh. In practice we have found that 100 iterations works well for our data, and has been used for all images. The smoothing is typically performed once at run-time, upon loading the data.

Although smoothed ambient occlusion is useful for selecting the scale of features that are rendered more opaquely, we also provide the ability to use the original ambient occlusion for color to retain detail (Figure 8).

4.3 Opacity Control

Arbitrary functions can be used to map the smoothed ambient occlusion values to opacity, however we desire a mapping that maintains the opacity of the inner cavity while providing control over the opacity of the outer surface. We experimented with functions such as *smoothstep*, however the following equation was determined via visual inspection to produce better results:

$$\alpha = \left(\frac{O_p}{\tau}\right)^{\rho}, \tag{4}$$

where α is clamped to $[0,1]$. The τ parameter provides a threshold such that an $O_p >= \tau$ gives an α of 1. A τ of 0.7 is used for all images in this paper (other than Figure 6, which uses a simple linear ramp). The ρ parameter controls the shape of the curve as an exponential, and in practice is allowed to vary over $[0,10]$. For a τ value of 1, a ρ value of 1 will give the same result as the simple linear opacity mapping decribed in section 4.1. Increasing ρ from 1 will reduce the opacity of the outer surface to a greater degree than the inner surface. Decreasing ρ from 1 will increase the overall opacity until ρ reaches 0, resulting in a constant α of 1 and a fully opaque surface. A graphical representation of Equation 4 is shown in Figure 7. The effect of changing ρ is shown in Figure 8, and is typically adjusted interactively by the user.

4.4 Supplementary Visualization Techniques

We also apply a number of supplementary visualization techniques to enable improved understanding of the enzyme active site cavity and ligand.

Silhouette-Edge Highlighting To maintain contextual information of the outer surface while rendering the interior surface more visible, silhouette-edge highlighting can optionally be applied:

$$\alpha = {\alpha_{in}}^{(\hat{N}\cdot\hat{E}+1)}, \tag{5}$$

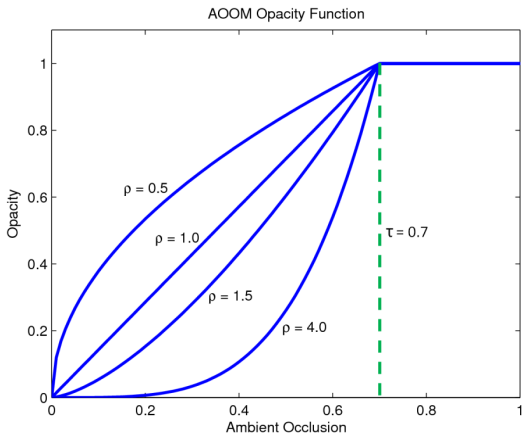


Figure 7: Example AOOM opacity functions.

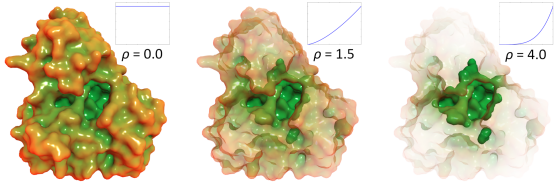


Figure 8: Result of changing the ρ parameter to adjust outer opacity while maintaining the opacity of the inner cavity.

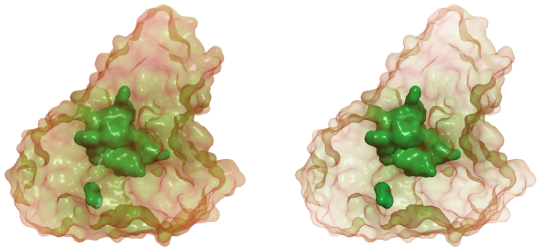


Figure 9: AOOM example without (left) and with (right) silhouette-edge highlighting.

where α_{in} is the opacity after applying Equation 4, and $\hat{N} \cdot \hat{E}$ is the dot product of the surface normal and the eye vector. This equation selectively reduces the opacity of areas on the surface with low opacity that face the viewer. Areas of high opacity are less affected, and areas of full opacity are unaffected. The calculation is performed in the same fragment program used for lighting and AOOM. Other edge highlighting techniques, such as suggestive contours [12, 13], could also be applied. Figure 9 shows the result of applying silhouette-edge highlighting.

Colored Surfaces To better understand the chemistry occuring in the active site, it can be useful to color the molecular surface by atom type. A nominal color coding is employed, with charged residues colored blue for cationic (positive) species and red for anionic (negative) species. The carbon backbone is colored green in our examples (coloring of specific groups, such as yellow sulfur groups, are also used). Because the molecular surface color conveys information, color mapping as described in Section 4.1 is not desirable in this case, as the nominal color encoding will be distorted. The silhouette-edge highlighting described in above can therefore be especially helpful when using such a color coding (Figure 10).

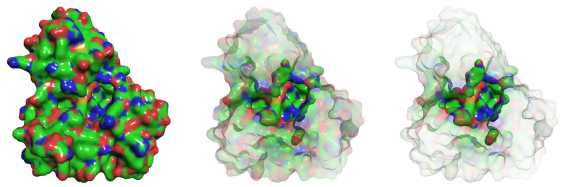


Figure 10: Colored molecular surface (left), along with AOOM renderings without (middle) and with (right) silhouette-edge highlighting.

Enclosed Region Removal Sometimes fully-enclosed regions are formed during the molecular surface calculation. These regions are not accessible from positions exterior to the enzyme (for molecules with a radius >= the surface probe radius) and can add visual clutter to the scene. These regions can be automatically removed by computing connected components on the molecular surface and rendering only the largest connected component, which will be the main molecular surface (Figure 1, Right vs. Figure 3, Top).

Textured Surfaces Textured surfaces can help improve surface-shape perception [21, 22]. We therefore apply an optional Perlin noise solid texture [31] to the molecular surface. This can be especially helpful when zoomed in close to the cavity surface (Figure 11).

Ligand Visualization To understand the interaction between enzymes and ligands, it can be useful to display the ligand within the active site of the enzyme. Figure 11 shows comparison views incorporating stick models of the ligand within the surface cavity, as well as specified enzyme residues of interest. The carbon backbones are colored grey. Other representations of the ligand and residues, such as spheres or van der Waals surfaces, could also be used.

Backface Opacity Modulation When displaying the ligand within the cavity, portions of the ligand can be obscured within small pockets of the cavity. To enable visualization of the ligand in these areas, further opacity modulation can be applied to render back-facing polygons more transparent (Figure 11). The opacity of back-facing polygons is modulated as:

$$\alpha = \alpha_{in} * \left(1.0 - (\hat{N} \cdot \hat{E}) * C\right), \qquad (6)$$

where α_{in} is the opacity after Equation 4 and optionally Equation 5 are applied, $\hat{N} \cdot \hat{E}$ is the dot product of the surface normal and the eye vector, and C controls the overall opacity reduction. This equation selectively reduces the opacity of back-facing surfaces more

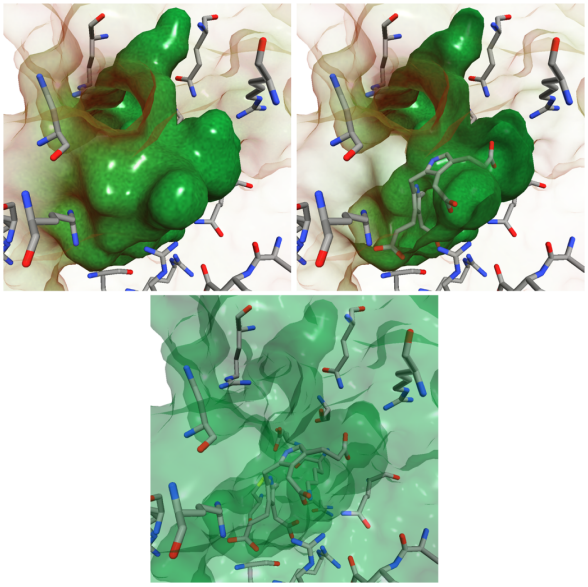


Figure 11: The top views show AOOM renderings of a closeup of the inner cavity. The image on the left shows the back-facing surface with full opacity, and the image on the right shows the back-facing surface rendered with a view-dependent opacity to reveal the ligand within. The bottom image shows the same viewpoint using standard transparency.

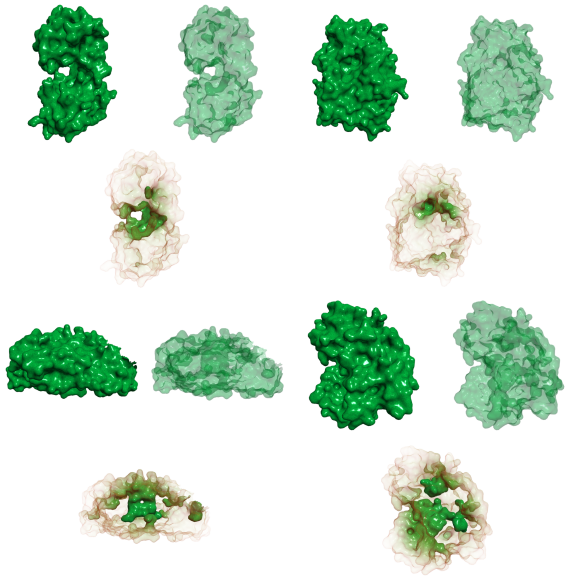


Figure 12: Various enzymes rendered using AOOM (bottom) to enable better perception of inner structure than with standard transparency (top right).

where the surface faces the viewer, providing subtle edge highlighting of the back-facing surface. For future work, it may be interesting to apply texture-based transparency methods to back-facing polygons to enable improved perception of the outer and inner surfaces of these pockets [24, 33, 44].

5 CONCLUSION AND FUTURE WORK

We have presented ambient occlusion opacity mapping (AOOM), a novel technique for viewing inner molecular surface structure. AOOM uses ambient occlusion information, a measure of the "hiddenness" of a particular point on a surface, to render occluded areas of a surface more opaque than non-occluded areas. We have shown the application of AOOM to the specific problem of visualizing the active sites of enzymes, as our collaborators have successfully used AOOM to better understand the structure of this inner cavity. Smoothing the ambient occlusion information over the surface enables control over the scale of the cavities to highlight. Color and opacity controls, including silhouette-edge highlighting, have also proven useful in highlighting the inner cavity of interest. AOOM is more effective than techniques such as transparency, clipping planes, and focal regions, and works for cases where cavity extraction techniques such as CASTp fail.

We have implemented AOOM via extensions to the Visualization Toolkit (VTK), and have created a test application using this code. Future work includes incorporating AOOM into existing molecular visualization packages, such as PyMOL, to take advantage of features, including measurements and ribbon-style rendering, that our collaborators already find useful.

It may also prove useful to apply AOOM in fields other than molecular visualization. Specifically, medical visualization and oil and gas visualization could benefit from AOOM, as both fields often work with data sets exhibiting inner structures of interest that may be occluded.

6 ACKNOWLEDGMENTS

We would like to thank Charles Lewis and Kenneth Ashe II from the Department of Biochemistry and Biophysics at the University of North Carolina at Chapel Hill for their help with this work.

REFERENCES

[1] Jean-Daniel Boissonnat, Olivier Devillers, and Jacqueline Duquesne. Computing Connolly surfaces. *J. Molecular Graphics*, 12(1):61–62, 1994.

[2] David Borland. *Flexible Occlusion Rendering for Improved Views of Three-Dimensional Medical Images*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2007.

[3] David Borland, John P. Clarke, Julia R. Fielding, and Russell M. Taylor II. Volumetric depth peeling for medical image display. In Robert F. Erbacher, Jonathan C. Roberts, Matti T. Gröhn, and Katy Börner, editors, *Visualization and Data Analysis, Proceedings of SPIE-IS&T Electronic Imaging 2006*, volume 6060, pages 35–45, January 2006.

[4] Rob Bredow. Renderman on film: Combining CG and live action, Course 16: RenderMan in Production. In *ACM SIGGRAPH 2002 Course Notes*, 2002.

[5] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and M. Eduard Gröller. Illustrative context-preserving exploration of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1559–1569, 2006.

[6] Ho-Lun Cheng and Xinwei Shi. Quality mesh generation for molecular skin surfaces using restricted union of balls. In *Proceedings of IEEE Visualization*, pages 399 – 405, 2005.

[7] Gregory Cipriano and Michael Gleicher. Molecular surface abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1608–1615, 2007.

[8] Michael L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 1983.

[9] Michael L. Connolly. The molecular surface package. *J. Molecular Graphics*, 11(2):139–141, 1993.

[10] Michael L. Connolly. Molecular surfaces: A review. *Network Science*, online article `http://www.netsci.org/Science/Compchem/feature14.htm`, 1996.

[11] Carlos Correa and Kwan-Liu Ma. The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1465–1472, 2009.

[12] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 15–24, 2004.

[13] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. In *Proceedings of SIGGRAPH*, pages 848–855, 2003.

[14] Paul J. Diefenbach. *Pipeline rendering: Interaction and realism through hardware-based multi-pass rendering*. PhD thesis, Department of Computer Science, University of Pennsylvania, 1996.

[15] Joachim Diepstraten, Daniel Weiskopf, and Thomas Ertl. Transparency in interactive technical illustration. *Computer Graphics Forum*, 21(3):317–325, 2002.

[16] Joachim Diepstraten, Daniel Weiskopf, and Thomas Ertl. Interactive cutaway illustrations. *Computer Graphics Forum*, 22(3):523–532, 2003.

[17] Joe Dundas, Zheng Ouyang, Jeffery Tseng, Andrew Binkowski, Yaron Turpaz, and Jie Liang. CASTp: Computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic Acids Research*, 34:W116–W118, 2006.

[18] Cass Everitt. Interactive order-independent transparency. Technical report, Nvidia Corporation, 2002.

[19] Jan Fischer, Dirk Bartz, and Wolfgang Strasser. Illustrative display of hidden iso-surface structures. In *Proceedings of IEEE Visualization*, pages 663–670, 2005.

[20] Athanasios Gaitatzes, Yiorgos Chrysanthou, and Georgios Papaioannou. Presampled visibility for ambient occlusion. *Journal of WSCG*, 16(1-3):17–24, 2008.

[21] James J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, 1950.

[22] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[23] David S. Goodsell. Visual methods from atoms to cells. *Structure*, 13(3):347–354, 2005.

[24] Victoria Interrante, Henry Fuchs, and Stephen M. Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):98–117, 1997.

[25] Adam G. Kirk and Okan Arikan. Real-time ambient occlusion for dynamic character skins. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, April 2007.

[26] Hayden Landis. Production-ready global illumination, Course 16: RenderMan in Production. In *ACM SIGGRAPH 2002 Course Notes*, 2002.

[27] Chang Ha Lee and Amitabh Varshney. Computing and displaying intermolecular negative volume for docking. In G. M. Nielson G.-P. Bonneau, T. Ertl, editor, *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Springer-Verlag, ISBN 3-540-26066-8, 2005.

[28] G. V. Louie, P. D. Brownlie, R. Lambert, J. B. Cooper, T. L. Blundell, S. P. Wood, M. J. Warren, S. C. Woodcock, and P. M. Jordan. PDB #: 1pda: Structure of porphobilinogen deaminase reveals a flexible multidomin polymerase with a single catalytic site. *Nature*, 359:33–39, 1992.

[29] Rakesh Mullicka, R. Nick Bryanb, and John Butmana. Confocal volume rendering: Fast segmentation-free visualization of internal structures. In Seong K. Mun, editor, *Medical Imaging 2000: Image Display and Visualization*. Proceedings of SPIE, SPIE Vol. 3976, 2000.

[30] Arthur J. Olson and Michael E. Pique. Visualizing the future of molecular graphics. *SAR and QSAR in Environmental Research*, 8(3-4):233–247, 1998.

[31] Ken Perlin. Improving noise. *Computer Graphics*, 35(3), 2002.

[32] Christof Rezk-Salama and Andreas Kolb. Opacity peeling for direct volume rendering. *Computer Graphics Forum*, 25(3):597–606, 2006.

[33] Penny Rheingans. Opacity-modulating triangular textures for irregular surfaces. In *Proceedings of IEEE Visualization*, pages 219–225, 1996.

[34] Frederic M. Richards. Areas, volumes, packing and protein structure. *Annual Review of Biophysics and Bioengineering*, 6:151–176, 1977.

[35] E. H. Rydberg, C. Li, R. Maurus, C. M. Overall, G. D. Brayer, and S. G. Withers. PDB #: 1kbb: Mechanistic analyses of catalysis in human pancreatic alpha-amylase: detailed kinetic and structural studies of mutants of three conserved carboxylic acids. *Biochemistry*, 41(13):4492–4502, April 2002.

[36] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Fast and robust computation of molecular surfaces. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 406 – 407, 1995.

[37] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 73–80, April 2007.

[38] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 97–105, 2007.

[39] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing to enhance real time molecular visualization. In *Proceedings of IEEE Visualization*, 2006.

[40] John Tate. Molecular visualization. In Philip E. Bourne and Helge Weissig, editors, *Structural Bioinformatics*, chapter 23. Wiley-Liss, 2003.

[41] Amitabh Varshney and Jr. Frederick P. Brooks. Fast analytical computation of richardï¿½s smooth molecular surface. In *Proceedings of IEEE Visualization*, pages 300–307, 1993.

[42] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization*, pages 139–146, 2004.

[43] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, second edition, 2004.

[44] Chris Weigle and Russell M. Taylor II. Visualizing intersecting surfaces with nested-surface techniques. In *Proceedings of IEEE Visualization*, pages 503–510, 2005.

Interactive Volume Rendering Aurora on the GPU

Orion Sky Lawlor* Jon Genetti†
Department of Computer Science, University of Alaska Fairbanks

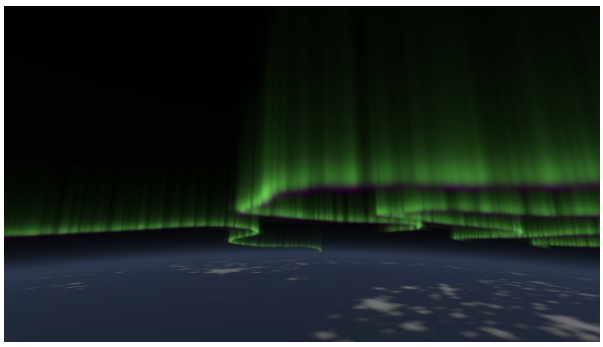


Figure 1: Our rendered aurora, 60km above Finland.

ABSTRACT

We present a combination of techniques to render the aurora borealis in real time on a modern graphics processing unit (GPU). Unlike the general 3D volume rendering problem, an auroral display is emissive and can be factored into a height-dependent energy deposition function, and a 2D electron flux map. We also present a GPU-friendly atmosphere model, which includes an integrable analytic approximation of the atmosphere's density along a ray. Together, these techniques enable a modern consumer graphics card to realistically render the aurora at 20–80fps, from any point of view either inside or outside the atmosphere.

Keywords: Volume rendering, aurora borealis, atmospheric scattering.

1 THE AURORA

The aurora borealis and aurora australis are beautiful phenomena that have fascinated viewers in Earth's polar regions for centuries. Aurora are generated when charged particles trapped by a planet's magnetic field collide with and excite gas in the upper atmosphere.

*e-mail:lawlor@alaska.edu
†e-mail:jdgenetti@alaska.edu

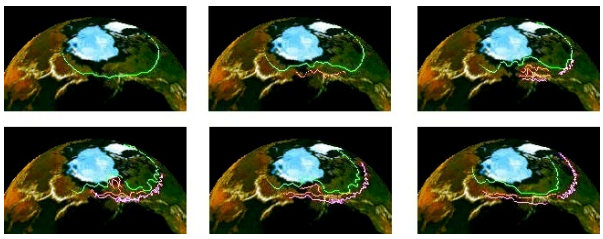


Figure 2: Global progress of a typical auroral substorm.

On Earth, these charged particles rarely penetrate below 50 kilometers altitude, and the aurora become difficult to discern above 500 kilometers due to the thin atmosphere.

The charged particle fluxes visible as auroral displays are driven by magnetohydrodynamics that are complex and the details are poorly understood, but the effects can be qualitatively described. As is typical in magnetohydrodynamics, magnetic effects expel currents from the body of a conductive plasma, compressing the charged particle currents flowing through the magnetosphere into thin sheets around one kilometer thick. As these current sheets are bent along magnetic field lines and intersect the atmosphere, they become visible as auroral "curtains," long linear stripe-like features. Depending on the activity level of the aurora, curtains can be nearly featureless greenish blur, or an extremely complex and jagged path.

A typical "auroral substorm" [Aka64] begins with simple, smooth curtains. These then grow and begin to fold over during substorm onset, resulting in many overlapping and interacting curtains, which become more and more complex and fragmentary as the substorm breaks up, and finally substorm recovery gives dim pulsating aurora. Recent work by Nishimura et al. [Nis10] has linked ground observations of pulsating aurora to space-based observations of electromagnetic waves deep in Earth's magnetotail, using the THEMIS satellites.

Because the detailed interactions of the charged particles and magnetic fields that drive auroral substorms are poorly understood, for rendering purposes we approximate their effect. We represent an auroral curtain's path using a time-dependant 2D spline curve "footprint," which are animated by hand to match the broad global outlines of an auroral substorm as it moves over the surface of the planet as shown in Figure 2.

1.1 Algorithm Overview

In this paper, we present a combination of techniques to interactively render the aurora on modern graphics hardware. To summarize our interactive GPU rendering algorithm:

1. We begin with aurora curtain footprints, described in Section 2, stored as 2D splines curving along the planet's surface.
2. We add 2D complexity to those curtain footprints by wrapping a long thin fluid dynamics simulation along them as described in Section 2.
3. We preprocess the curtain footprints into a 2D distance field described in Section 3.2, and stored in another GPU 2D texture and used to accelerate rendering.
4. We stretch the curtains into 3D using an atmospheric electron deposition function, as described in Section 2.1. The deposition function is expensive and constant, so it is stored as a GPU texture lookup table.
5. For each frame, we shoot rays from the camera through each pixel onscreen. Any camera model may be used.
6. For each ray, we determine the portion of the ray that intersects the aurora layer and atmosphere, and determine the layer compositing order as described in Section 3.1.
7. To intersect a ray with an aurora layer, we step along the ray at conservative distances read from the distance field, as described in Section 3.2. At each 3D sample point, we sum up the auroral emission as the product of the 2D curtain footprint and the vertical deposition function.
8. To intersect a ray with the lower atmosphere, we evaluate a closed-form airmass approximation as described in Appendix A.
9. Final displayed pixels are produced by compositing together the resulting aurora, atmosphere, and planet colors followed by an sRGB gamma correction, as described in Section 3.3.

Section 4 describes the performance of our algorithm on various graphics hardware.

2 MODELING THE AURORA IN 3D

Because curtains become fragmented and complex during the highly excited periods of an auroral substorm, splines alone do not convey the complexity of real curtains, as illustrated in Figures 3 and 4. Several approaches have been used to simulate this complexity,

Figure 3: Photograph of auroral curtains during a moderate substorm. The shutter was open for four seconds.

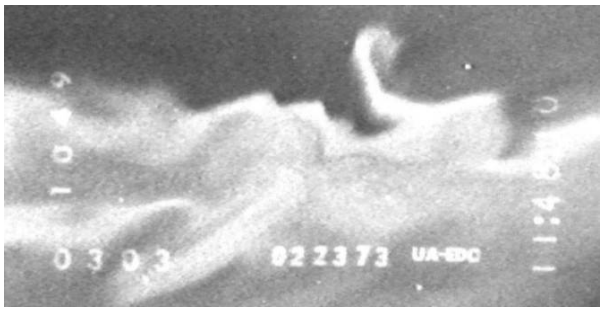


Figure 4: High-speed video of a portion of a very active curtain. Field of view is 4km wide.

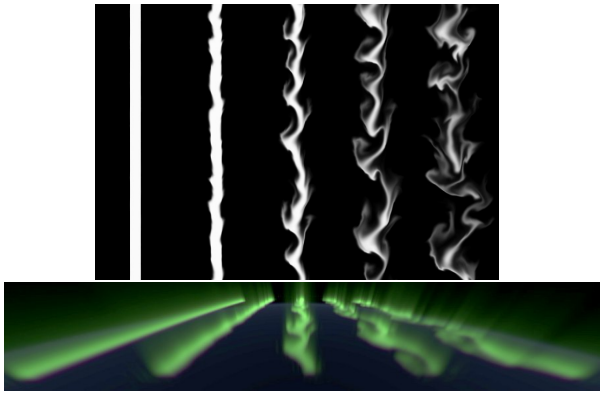


Figure 5: Portions of the 2D fluid dynamics simulations we use to model small-scale curtain complexity, and the resulting 3D auroral curtains.

such as raycasting caustics, but we find the phenomena are better matched by a fluid dynamics simulation.

To simulate aurora curtain footprint complexity, we use an simple 2D Stam-type [Sta99] fluid advection simulator. We use a multigrid divergence correction approach for the Poisson step, which is both asymptotically faster than an FFT or conjugate gradient approach, and makes the simulator amenable to a graphics hardware implementation. The simulator is solving a Kelvin-Helmholtz instability problem, with the fluid shear zone lying along the flux center of the auroral curtain, as illustrated in Figure 5. We perform the simula-

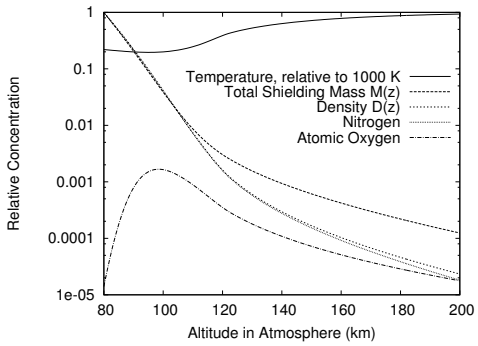


Figure 6: MSIS atmosphere as a function of altitude.

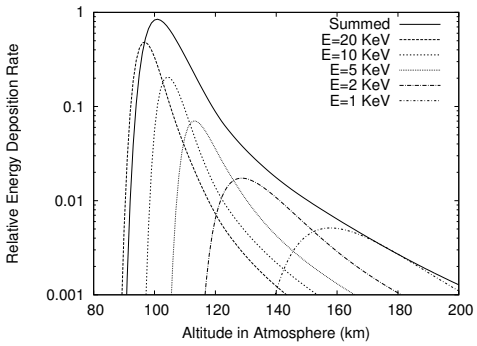


Figure 7: Auroral energy as a function of altitude.

tion in a long vertical domain with periodic boundary conditions, so we could replicate the same simulation along an arbitrarily long spline. The resulting simulated auroral curtain is stretched along the spline that defines the center line of the curtain. For quiet early periods during the substorm, we use the initial steps of the simulation, before substantial turbulence has distorted the smooth initial conditions; later more chaotic curtains are represented using later steps in the simulation, when the simulation's fluid turbulence results in a very complex electron flux pattern. The magnetosphere's actual plasma dynamics are of course very different from simple Navier-Stokes fluids, but this simulation seems to approximate the final turbulent appearance of the aurora reasonably well.

2.1 Aurora Vertical Deposition

We use splines to impose the global location of the auroral curtains, and fluid dynamics to approximate the small-scale variations in brightness, but both of these give only an electron flux footprint on the surface of the planet, in 2D. To create a full 3D volume model of the aurora, we must specify how the electrons are deposited through the atmosphere, via an electron deposition function.

The depth that charged particles penetrate the atmosphere depends on both the velocity of the charged particles and the atmosphere's state. However, the state of the upper atmosphere is not constant, due to variable energy input from solar radiation, ground-based upward travelling radiation, and even variable auroral energy deposition itself. Since the auroral energy deposition profile depends on a variety of factors, including feedback due to auroral heating, an exact deposition model would require us to simulate the spatial and temporal variations in the upper atmosphere's density, temperature, and chemistry. Software exists to do this, such as NCAR's thermospheric general circulation model, but it is not amenable to either the GPU or to realtime interactive simulation. Instead, we begin with the standard MSIS-E-90 atmosphere [Hed91], as shown in Figure 6.

We then apply the Lazarev charged particle energy deposition model [Lum92], which is still the definitive model for low-energy auroral electrons [Fan08]. The inputs to the Lazarev model are the particle energy E and the atmosphere's mass M_z and density D_z at the desired height z, and the output is the auroral energy deposition rate A_z:

E Initial energy of incoming particles, in thousands of electron-volts [keV]. For aurora, this is 1–30keV [Fan08]. These equations work well below 32keV.

z Altitude above surface to evaluate deposition.

D_z Atmosphere's density at altitude z [g/cm^3]. This is listed directly in the MSIS data.

$M_z = \int_z^\infty D_{z'} dz'$ Atmosphere's total shielding mass above altitude z [g/cm^2].

$M_E = 4.6 \times 10^{-6}\, E^{1.65}$ Characteristic shielding mass for particles of energy E [g/cm^2]

$r = M_z / M_E$ Relative penetration depth [unitless]

$L = 4.2\, r\, e^{-r^2 - r} + 0.48\, e^{-17.4\, r^{1.37}}$ Lazarev's unscaled interaction rate [unitless]

$A_z = L\, E\, (D_z / M_E)$ Aurora energy deposition rate at altitude z.

The result of the Lazarev deposition model is shown in Figure 7 for several discrete input energies, as well as a sum over energies from 1keV to 20keV. Various parameterizations of this deposition function exist, such as the popular Thermospheric General Circulation Model [Rob87], which assumes a Maxwellian distribution of electron energies. TGCM is actually simple enough to evaluate per pixel at runtime, as we explore in Section 4. However, both the Lazarev or TGCM models need an atmosphere model as input, and the thermosphere's density profile D_z is complex, as shown in Figure 6. Since we will need a lookup table to store the atmosphere's shielding mass and density, we simply pre-evaluate the deposition function for various energies and store the result in a table.

2.2 Prior Work in Aurora Modeling

We extend the excellent and rigorous aurora rendering work of Baranoski et al. [Bar00] in several ways. First, this prior work forward maps aurora curtain points onscreen followed by a guassian blur, while our renderer walks backward along camera rays accumulating visible energy. Our raytracing approach allows us to render to arbitrary resolutions and produce sharp rendered images. Second, we provide an interactive GPU implementation which includes the effect of the lower atmosphere on the aurora and allows us to render the aurora from any point inside or outside the atmosphere. In the prior work, electron-atmosphere impacts are simulated explicitly, while we simply look up their well known altitude dependent statistical energy deposition function. Finally, the prior work's curtains are constructed from a combination of sine wave with phase shift oscillations and a caustic-type electron beam deflection model; while our curtains begin as splines, with smaller turbulent deflections applied via a fluid dynamics simulation.

The later work of Baranoski et al. [Bar05] presents a detailed physically plausible model of the magnetohydrodynamics of a charge sheet's path through the magnetosphere prior to becoming visible as an auroral curtain. There appears to be an almost exact analogy between this work and our fluid dynamics simulation of curtain dynamics: electric charges with inertia interact via an electrostatic field, while fluid parcels with inertia interact via a pressure field. Both electrodynamic and fluid dynamic simulations use a multigrid Poisson solver to control field divergence, and the results appear roughly similar as well. One difference is we have not yet attempted to specialize our initial conditions to generate the spiral structures visible as auroral surges.

3 GPU RAYTRACING THE AURORA

Raytracing is a rendering technique that finds a scene's color along a ray by intersecting the ray with the scene geometry. Raytracing is computationally demanding, and the first interactive raytracers used a combination of carefully constructed scenes (such as a set of spheres) and massive parallel computing horsepower. University of Utah researchers [Par98] used a large shared-memory machine for this, while John Stone's Tachyon [Sto98] used a network of distributed-memory workstations. GPU raytracing is such a natural fit that initial work in this area [Pur02] actually preceded fully programmable GPU hardware, and an abundance of modern work exists. Similarly, volume rendering via raytracing is a venerable and well known technique [Kaj84].

3.1 Aurora rendering geometry

The aurora are almost perfectly emissive phenomena, since the degree of absorption and scattering by the atmosphere is vanishingly small around 100km altitude. Even at sea level air's optical properties are reasonably close to that of vacuum, and at 100km altitude the air's density is a millionfold smaller. The isotropic emissions, and lack of absorption and scattering, simplifies Kajiya's rendering equation [Kaj84] for the aurora layer into a single integral along the path of the ray.

Since aurora only appear in the upper layers of the atmosphere, we can treat them as a separate purely emissive "aurora layer." Below 80km is the bulk of the lower atmosphere, which both absorbs and scatters light as discussed in Appendix A. Underneath all of this is the planet's surface. Because the lower atmosphere includes scattering, implemented using alpha blending, we must composite the layers in the correct order.

A general-purpose raytracer typically uses recursion to resolve the depth order of multiple layers of translucent geometry that intersect a ray, but this general solution is not appropriate in our case. First, GPU hardware that directly supports recursion was only introduced in 2010 with the NVIDIA Fermi line, and most current cards do not directly support recursion. Second, even where it is supported this recursive search for geometry is expensive, typically requiring $O(n^2)$ intersection tests to determine the depth order of n translucent layers, and we find the many branches required can become a limiting factor in a high performance GPU raytracer.

Thus instead of a recursive search, for each ray we programmatically determine the correct compositing order of the intersected geometry, as summarized in Figure 8. The easy case is 8(a), where the ray misses all geometry and heads out into deep space. Case 8(b) is a ray that enters the aurora layer, accumulates some emitted energy, and exits. The most complex case is 8(c), where the aurora layer is entered twice: once before the atmosphere, then some aurora light is scattered out by the atmosphere, and finally a disjoint stretch of aurora layer emits more light into the ray. Finally, case 8(d) begins on the planet's surface, whose light is attenuated by the atmosphere, and then some aurora light is picked up before reaching the viewer. The same cases apply for a viewer inside the aurora layer. For a viewer inside the lower atmosphere, the only two compositing possibilities are atmosphere then planet, or atmosphere then aurora.

One limitation of our explicit ray compositing order is we do not support atmospheric refraction. However, Earth's atmosphere only very gently refracts rays, resulting in a maximum curvature near the horizon which is less than 1/6 of the planet's curvature, so we feel it is acceptable to ignore atmospheric refraction.

Given a portion of a ray that intersects the aurora layer, in principle we step through the layer accumulating aurora energy, at each step sampling the aurora curtain footprint in 2D and multiplying it by the height-dependent energy deposition function. The step size is

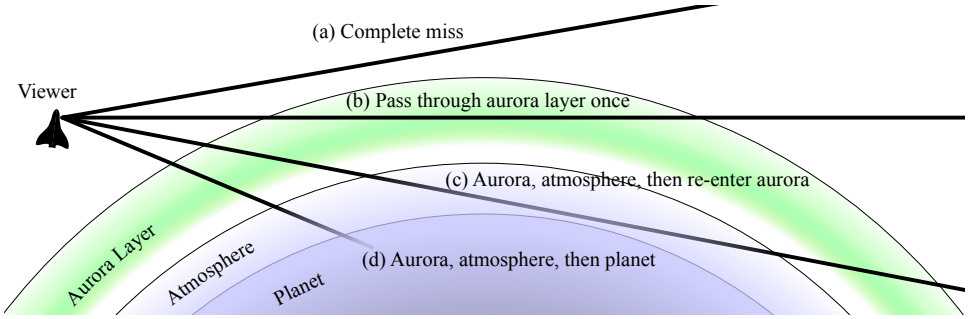


Figure 8: Possible ray/geometry intersection paths for camera rays originating outside the atmosphere.

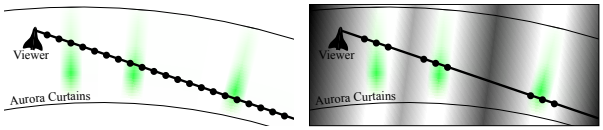


Figure 9: Naive ray stepping, left, is inefficient when curtains are sparse. Using a distance field, as shown on the right, allows the raytracer to take much larger steps in the empty spaces between curtains.

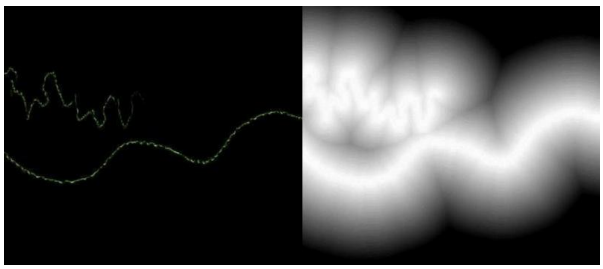


Figure 10: On the left, aurora curtain footprints. On the right, the distance field to accelerate raytracing those curtains.

a tunable parameter, with finer steps giving more aurora detail but as we show in Section 4, taking more time to compute. The step size is limited by the resolution of the aurora footprint texture: an 8192x8192 aurora footprint stretched across a 12742km diameter planet gives pixels that are 1.55km along the coordinate axes, and 2.2km diagonally. We find a 2km step size gives a reasonable quality image, but with naive sampling is quite slow. In the next section, we show how to accelerate the aurora sampling process.

3.2 Acceleration via a Distance Field

The auroral layer is hundreds of kilometers high, and wraps around a planet thousands of kilometers in diameter. Yet auroral curtains are only a few kilometers thick, so as we step along a ray we must sample the aurora layer at least every few kilometers to avoid missing curtains. Even modern GPU hardware cannot support thousands of such 3D samples per pixel in real time, since there are millions of onscreen pixels.

However, most of the auroral layer does not actually contain curtains, so if we could skip over the empty space between curtains, we could dramatically improve our overall performance. Figure 9 illustrates the problem, and the solution we use: a distance field [Coh94]. This field stores the distance to the nearest geometry, which allows the raytracer to take much larger steps through empty space.

The distance field is stored as a 2D texture, with a slightly lower resolution than the aurora curtain image. As we step along a ray, we read the step size from the distance field, so we step at a fine 2km/step rate while inside curtains; yet can take much longer steps far from curtains, up to 1000km/step, without ever skipping over a curtain. In pseudocode, our sampling loop through the aurora is as follows.

```
float t = ray.start;
while (t < ray.end) {
    vec3 P = ray.origin + ray.dir*t;
    t += distance_field(P);
    aurora += sample_aurora(P);
}
```

One surprising aspect of the GPU branch hardware is that it is actually a performance loss to skip the aurora sampling when distant from a curtain. We found it to be at least 18% slower to do the following "optimized" sampling; our other attempts at similar optimizations have been up to sevenfold slower!

```
float t = ray.start;
while (t < ray.end) {
    vec3 P = ray.origin + ray.dir*t;
    float d = distance_field(P);
    t += d;
    if (d<ε) /* inside curtain */
        aurora += sample_aurora(P);
}
```

The performance problem in this sort of loop is branch divergence, when some GPU threads take the distance-dependent branch and sample the curtain while others do not. The large GPU branch divergence penalty exceeds the savings from avoided samples, which makes

it faster to simply sample everywhere than to carefully decide whether to sample or not.

We generate the distance field from the curtain image on the GPU, but as a preprocess before rendering. We use a clever constant-time algorithm known as "jump flooding" [Ron06], which takes distance propagation steps at power of two distances to fill the distance field across the 2D image.

3.3 Coloring the Aurora

On short timescales, the upper layers of the aurora are green, while the lower layers have a purple tinge. We use the Baranoski et al. [Bar00] approach to convert the auroral emissions' isolated spectral color peaks to CIE XYZ and then a linear sRGB colorspace.

More difficult are numerical problems encountered while summing thousands of dim samples. In Baranoski et al., aurora samples are forward mapped and summed in a framebuffer, while we step along camera rays in a loop on the GPU. Because the GPU registers are floating-point, and floating-point framebuffers are expensive, a raytracer can more efficiently sum aurora samples in a high precision and high dynamic range linear colorspace. We then convert to the standard sRGB gamma of 2.2 using the following function, which outputs a color with vector magnitude equal to the old magnitude raised to the 1/2.2 power.

```
float brightness=length(color);
return color*pow(brightness,1/2.2-1);
```

4 PERFORMANCE ANALYSIS

We use the standard OpenGL Shading Language, GLSL, to implement our GPU aurora raytracer. Unlike the general-purpose GPU languages CUDA and OpenCL, the older GLSL is specialized for rendering tasks, so it directly supports graphics hardware features such as anisotropic mipmapping. Recent work on VOREEN [Men10] showed CUDA only improves performance when volume samples overlap, such as in gradient calculations. Table 1 compares the performance of our GPU aurora rendering algorithm across various GPU families, and a C++ OpenMP multicore CPU version of the algorithm. Even using four cores and nearest-neighbor texture sampling, the CPU runs about a hundred times slower than the GPU versions.

Table 2 lists the performance impact of various algorithm and parameter modifications. This is a list of alternatives not chosen for the current implementation, although many of these could still be useful.

Our raytracer acceleration distance field results in rather dramatic per-pixel performance variations, as shown in Figure 11. The corresponding frame is shown in Figure 1. Where multiple curtains cross camera rays the rendering cost can be hundreds of nanoseconds per pixel, while empty regions of space require less than

GPU	FPS
NVIDIA GeForce GTX 280	60fps
NVIDIA GeForce 8800M GTS	38fps
ATI Radeon HD 4830	23fps
Intel Q6600 2.4GHz Quad-Core CPU	0.4fps

Table 1: Comparing renderer performance across hardware. Resolution is 720p: 1280x720.

Modified Rendering Method	Cost
No distance field, use naive stepping	+350%
Make aurora layer 100km thicker	+32%
Take 1km steps through aurora, not 2km	+60%
Take 4km steps through aurora, not 2km	-33%
No table, use TGCM deposition function	+55%
No decibel map, linear deposition table	-10%
No deposition function, constant value	-14%
No curtain footprint image lookup	-14%
No exponential atmosphere	-15%
No planet texture	-0.6%
No sRGB gamma correction	-0.5%

Table 2: Performance impact of various alternatives. Positive time cost lowers framerate.

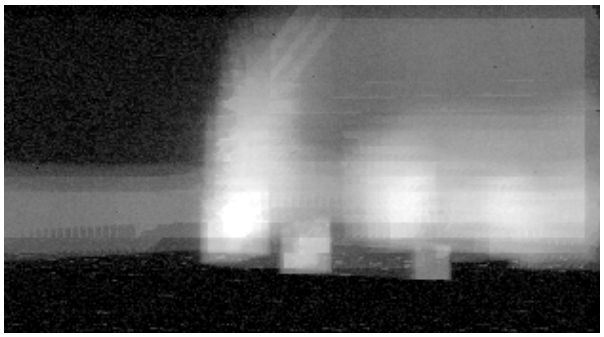


Figure 11: Measured rendering time per pixel: black represents 10ns/pixel, white represents 200ns/pixel.

ten nanoseconds per pixel. This experiment was run on the NVIDIA GeForce 8800M GTS; timings on different cards vary, but the ratios are similar. This figure is somewhat blurred due to the nature of GPU performance analysis: GPU hardware provides no means to time individual pixels, and in fact extensive GPU pipelining makes per-pixel timing difficult to even define, so instead we time overlapping blocks of 64x64 pixels. After several repetitions, the median per-block times are converted to per-pixel times by subtracting off the per-block overhead and dividing by the number of pixels. The remaining sampling jitter due to OS and driver overhead is approximately $\sigma = 2$ns/pixel.

4.1 GPU Aurora on a Powerwall

We used the parallelizing library MPIglut [Law08] to port our sequential OpenGL/GLUT aurora rendering application to a twenty-screen powerwall, as shown in

Figure 12: Interactive aurora rendering on a powerwall cluster with ten GPUs and twenty screens at 29fps.

# GPUs	Resolution	FPS	Speedup
1	1680x2100	35	1
2	3360x2100	30	1.6
4	6720x2100	27	3.0
8	6720x4200	29	6.5
10	8400x4200	29	8.2

Table 3: Parallel aurora rendering via MPIglut.

Figure 12. This was a surprisingly straightforward process, involving recompiling the rendering application with MPIglut instead of glut, and running the resulting binary. Scalability as shown in Table 3 is reasonably good, although view-dependent load imbalance becomes large when some screens must draw complex curtains and other screens only empty space; for the benchmark this impacts the two and four GPU values somewhat. The aggregate rendering rate on ten NVIDIA GeForce GTX 280 cards is a little over 29 frames per second at 8400x4200 resolution, or just over a billion finished pixels per second.

5 CONCLUSIONS

With only moderate programming effort, modern graphics hardware is capable of truly incredible amounts of computation. We have harnessed that power to render the aurora at interactive rates, but much work remains.

At the moment, our raytracer implementation stands alone, and includes no polygonal geometry. It would be relatively straightforward to extend this to a hybrid raytracer, where ordinary polygon-based geometry is first rasterized to a typical depth buffer, and these depth values are then used to limit the extent of each ray [Sch05]. This extension would allow the techniques described in this paper to add atmospheric and aurora effects to a scene that includes terrain, vegetation, spacecraft, or other geometry.

We currently render a single instantaneous snapshot of the aurora; the viewer is free to move, but the curtains are stationary. It should be straightforward to extend this to animating curtains, and we have done so offline, but image I/O and texture upload rate becomes an issue when rendering in realtime. Similarly, we currently do not integrate the curtains across the minutes-long timescale that gives high red aurora. This should be a simple change to our input curtain footprint images. Both changes should allow a detailed comparison with the widespread seconds-long-exposure photographic images of the aurora.

Since aurora are purely emissive phenomena, our atmospheric airmass model currently ignores clouds and the interesting multiple scattering effects of sunlight on the air. Incorporating these effects would allow us to simulate aurora at sunrise, or aurora rising over a thunderhead. More ambitiously, implementing a global illumination algorithm such as photon mapping or path tracing could allow aurora to cast light onto complex geometry, such as a mountainside or spacecraft.

Aurora are visible on many planets, and often display curtains and dynamics similar to those on Earth. However, the dynamics of aurora on planets without a single dominant magnetic field, such as Venus or Mars, can be quite different, and simulations would be beneficial for studying these fascinating phenomena.

ACKNOWLEDGEMENTS

The authors sincerely thank Dr. Syun-Ichi Akasofu for providing the schematics of a typical auroral substorm and the video capture in Figure 4, as well as Dr. Bill Brody for digitizing and animating that substorm as a series of continuous splines as shown in Figure 2. Our night earth texture is from NASA's Visible Earth project. Previous support for this project has been provided by the American Museum of Natural History.

REFERENCES

[Aka64] Syun-Ichi Akasofu. The development of the auroral substorm. *Planetary and Space Science*, 12(4):273 – 282, 1964.

[Bar00] G.V.G. Baranoski, J.G. Rokne, P. Shirley, T. Trondsen, and R. Bastos. Simulating the aurora borealis. In *Computer Graphics and Applications*, pages 422–432, october 2000.

[Bar05] Gladimir V. G. Baranoski, Justin Wan, Jon G. Rokne, and Ian Bell. Simulating the dynamics of auroral phenomena. *ACM Trans. Graph.*, 24(1):37–59, 2005.

[Coh94] D Cohen and Z Sheffer. Proximity clouds–an acceleration technique for 3D grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[Fan08] X. Fang, C. E. Randall, D. Lummerzheim, S. C. Solomon, M. J. Mills, D. R. Marsh, C. H. Jackman, W. Wang, and G. Lu. Electron impact

ionization: A new parameterization for 100 eV to 1 MeV electrons. *J. Geophys. Res.*, 113(A09311), 2008.

[Hed91] A. E. Hedin. Extension of the MSIS Thermosphere Model into the Middle and Lower Atmosphere. *J. Geophys. Res.*, 96(A2):1159–1172, 1991.

[Kaj84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, 1984.

[Law08] Orion Sky Lawlor, Matthew Page, and Jon Genetti. MPIglut: Powerwall programming made easier. *Journal of WSCG*, pages 130–137, February 2008.

[Lum92] D. Lummerzheim. Comparison of energy dissipation functions for high energy auroral electrons and ion precipitation. Technical Report UAG-R-318, Geophys. Inst., Univ. of Alaska-Fairbanks, April 1992.

[Men10] Jörg Mensmann, Timo Ropinski, and Klaus H. Hinrichs. An advanced volume raycasting technique using GPU stream processing. In *GRAPP Proceedings*, pages 190–198, 2010.

[Nis10] Y. Nishimura, J. Bortnik, W. Li, R. M. Thorne, L. R. Lyons, V. Angelopoulos, S. B. Mende, J. W. Bonnell, O. Le Contel, C. Cully, R. Ergun, and U. Auster. Identifying the driver of pulsating aurora. *Science*, pages 81–84, October 2010.

[Par98] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive ray tracing for isosurface rendering. *Visualization Conference, IEEE*, 0:233, 1998.

[Pur02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002.

[Rob87] R. G. Roble and E. C. Ridley. An auroral model for the NCAR thermospheric general circulation model (TGCM). *Annales Geophysicae, Series A - Upper Atmosphere and Space Sciences*, pages 369–382, december 1987.

[Ron06] Guodong Rong and Tiow-Seng Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, New York, NY, USA, 2006. ACM.

[Sch05] Henning Scharsach. Advanced GPU raycasting. In *Proceedings of CESCG*, 2005.

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99 Conference Proceedings*, pages 121–128, 1999.

[Sto98] John Stone. An efficient library for parallel ray tracing and animation. Master's thesis, Dept. of Computer Science, University of Missouri Rolla, 1998. http://jedi.ks.uiuc.edu/~johns/.

[You69] A.T. Young. High-resolution photometry of a thin planetary atmosphere. *Icarus*, 11(1):1–23, March 1969.

A CALCULATING AIRMASS

The integral of atmospheric density along a ray, known as "airmass," is widely used in astronomy, and we use it to approximate both the aurora light lost to the atmosphere, and night sky light added. A gravitationally bound atmosphere of uniform temperature and composition falls off in density at an exponential rate with height: $D(z) = e^{-z/H}$, with the exponential constant H known as the atmosphere's "scale height." The airmass integral along a ray parameterized by t is then:

$$A = \int_{t_s}^{t_e} D(z(t))dt = \int_{t_s}^{t_e} e^{-z(t)/H}dt$$

Even assuming a spherical planet, height varies nonlinearly along the ray path: $z(t) = \text{length}(\vec{S}+t\vec{D}) - r = \sqrt{a+bt+ct^2} - r$, so:

$$A = \int_{t_s}^{t_e} e^{-\frac{\sqrt{a+bt+ct^2}-r}{H}}dt$$

This integral cannot be solved in closed form. A trigonometric substitution [You69] allows high-order terms to be discarded, giving an integral that is easy to evaluate at the surface of the planet or at infinity, but a general raytracer requires arbitrary start and end points. We do this by approximating $z(t)/H$ with a quadratic $m + lt + kt^2$. We can eliminate the linear term l by translating the ray parameter t to t', leaving m as the height of closest approach of the ray to the planet, and k as the quadratic slope of that approach, both measured in scale height units.

$$A \approx \int_{t'_s}^{t'_e} e^{-m-kt^2}dt$$

This integral can be evaluated exactly using the error function "erf":

$$A \approx e^{-m}\sqrt{\frac{\pi}{4k}}\left(\text{erf}(\sqrt{k}t'_e) - \text{erf}(\sqrt{k}t'_s)\right)$$

Some GPU languages like GLSL do not have a built-in erf, so we use the Winitzki approximation:

$$\text{erf}(x) \approx \sqrt{1 - e^{-x^2\frac{\frac{4}{\pi}+0.147x^2}{1+0.147x^2}}}$$

Despite the plentiful transcendentals, this performs quite well on the graphics card at runtime. Despite the stacked approximations, accuracy appears quite good as well, except where numerical roundoff causes the erf difference to approach zero. This case can be handled by either falling back to a linear approximation of $z(t)$, or by interpreting the finite difference of erf values as a scaled derivative of erf: e^{-kt^2}.

Fast and Memory Efficient Feature Detection using Multiresolution Probabilistic Boosting Trees

Florian Schulze
VRVis Center for Virtual Reality and Visualization Research
fschulze@vrvis.at

David Major
VRVis Center for Virtual Reality and Visualization Research
dmajor@vrvis.at

Katja Bühler
VRVis Center for Virtual Reality and Visualization Research
buehler@vrvis.at

Abstract

This paper presents a highly optimized algorithm for fast feature detection in 3D volumes. Rapid detection of structures and landmarks in medical 3D image data is a key component for many medical applications. To obtain a fast and memory efficient classifier, we introduce probabilistic boosting trees (PBT) with partial cascading and classifier sorting. The extended PBT is integrated into a multiresolution scheme, in order to improve performance and works on block cache data structure which optimizes the memory footprint. We tested our framework on real world clinical datasets and showed that classical PBT can be significantly speeded up even in an environment with limited memory resources using the proposed optimizations.

Keywords: Feature Detection, Machine Learning, Decision Trees

1 INTRODUCTION

In the past years various methods for automatic processing and understanding of medical 3D image data have been developed. One important building block is the automatic detection of anatomical landmarks. Detection of these features stands often at the beginning of the processing pipeline: it transforms the dense volume representation into a sparse set of possible landmark locations, allowing a significant acceleration of subsequent high level segmentation methods.

However, making the transition from pure research algorithms which focus often solely on detection performance to real world radiology applications brings a number of additional requirements into consideration. The algorithm has to be able to deal with possibly limited technical resources - not all workstations in a hospital might be equipped with the newest hardware, and the algorithm shall run in the context of radiology workstation software which already occupies resources. Excellent time performance is required because automatic algorithms often substitute manual workflows while the result must be authorized and/or adjusted by the radiologist. In this case an automatic algorithm will only be used if the execution time of the algorithm is considerably shorter than the manual approach would be.

This work presents a highly optimized general purpose feature detection framework for the effective reduction of possible feature candidate positions in 3D image data as preprocessing step for more expensive object detection methods. Referring to the clinical application context, we designed our method according to the following requirements:

1. Time Performance: The result must be calculated in relatively short time (e.g. within seconds) in order to be usable in a clinical environment.

2. Memory Performance: The algorithm must also execute on standard PCs with limited technical resources.

Thus, the focus of the proposed algorithm and its implementation is on a small adaptable memory footprint while retaining as much execution speed as possible.

2 RELATED WORK

Object recognition, and local feature detection as a subdiscipline of it, are since many years core topics of computer vision research.

Point based methods beginning with the Harris corner detector [HS88] try to automatically extract points of interest from an image. Exact control of which points are extracted is not supported, therefore recognition of complex structures/areas is done by combining sets of feature points. The most prominent point detector is the SIFT algorithm [Low99] which overcomes the limitations of previous solutions by being scale, rotation and perspective invariant. However, translating SIFT, which is aimed for 2D images, to 3D volumes suffers from dramatic performance problems. Niemeijer et al. report in [NGL+09] that SIFT feature extraction on a

$200 \times 200 \times 1024$ volume downscaled by 50% takes 10 minutes to compute.

Machine learning based approaches use a (learned) classifier to decide if a specific region of an image belongs to an object. A prominent example for this class of algorithms is the method for real time face detection presented by Viola and Jones [VJ01] that uses a cascade of boosted weak classifiers. A more general approach has been proposed by Tu et al. [Tu05] by introducing Probabilistic Boosting Trees (PBT). PBTs are decision trees which use boosted learners as classifiers in each tree node. Violas boosted classifier cascades are a special case of a PBT. An alternative to PBTs is the popular d-tree forests method [MDUA07] which produces higher detection rates, but with the drawback of much higher execution costs [LK08].

PBTs have been successfully applied on tissue classification on medical images: Militzer and Vega-Higura [MV09] use PBT for bone removal in CT angiography. The volume is first split into segments using the watershed algorithm, then each segment is classified with PBT.

Fast preselection of feature candidates for more expensive high level methods is the topic of the paper of Langer and Kuhnert [LK08]. They integrate classical decision trees with simple color based features and a multiresolution scheme for candidate computation for the expensive SIFT feature detection.

The problem of the large memory footprint of volume data is often discussed in context of volume rendering. LaMar et al. [LHJ99] use an octree structure with blocks containing different resolutions, where only the needed subvolume is downloaded to graphics hardware. However, the whole volume data still has to fit into main memory. This has been improved by Guthe et al. [GWGS02] who proposed to hold the data 30:1 wavelet compressed in memory and extract needed data on demand block-wise and cache the data as long as possible.

The purpose of our feature detection method is similar to that of Langer and Kuhnert [LK08] since we also aim to reduce the list of possible candidate position as much as possible for later more expensive methods. Langer and Kuhnert tailored their algorithm especially for pre-filtering for SIFT feature computation. In difference to them we decided to use the more general PBT [Tu05]. This has several advantages: first, it is independent from SIFT features and easily adaptable to any kind of landmark/structure. Second, decision tree methods can capture large image variabilities while only need to execute $\log n$ weak classifiers. Third, they are robust against over-fitting unlike classic decision algorithms.

Our contribution. To satisfy the high performance requirements to the algorithm in a clinical environment, we extend the original PBT by integrating cascading tree nodes into normal tree building and introduce the concept of classifier sorting (Section 3.1). Both result in higher execution speed of the classifier. A second performance optimization is achieved by integrating the PBT into a multiresolution classification scheme (Section 3.2). An effective postprocessing step is introduced that applies particle filters to compute probability maps for candidate features for outlier detection (Section 3.3). The memory footprint of our feature detection framework is optimized by the introduction of a multiresolution, multi-derivative block cache data structure (Section 4). The performance of our method has been evaluated on a real world clinical usecase (Section 5).

3 ALGORITHM

In the following we explain in detail the classifier and our extensions on it (Section 3.1), the multiresolution feature detection framework (Section 3.2) and the postprocessing step based on candidate probability (Section 3.3).

3.1 Probabilistic Boosting Tree with Partial Cascading and Classifier Sorting

Probabilistic Boosting Trees. A PBT [Tu05] is a special kind of decision tree which holds at each tree node a boosted classifier. PBTs are trained top down. Based on a set of positive and negative samples a boosted classifier with a limited number of weak classifiers is trained for each tree node. On each recursion level the sample set is split using the generated classifier and the new subsets are used to train positive and negative child branches. Although multi-class classifiers are possible, we limited our implementation to the simple two-class model.

Classical Cascading. If the boosted classifier in each tree node is trained in a way that it does not produce false negative results, the resulting decision tree consists of positive child nodes only. Traversing this tree has only one sequential path and degenerates to the cascade of boosted classifiers of Viola and Jones [VJ01] (see figure 1 left). Cascading improves execution speed. It allows the classifier to early terminate and reduces

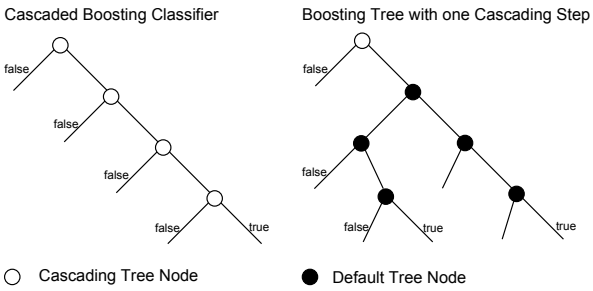


Figure 1: Probabilistic Boosting Tree. Left, tree with cascading nodes only. Right, one cascading node at the tree root followed by a default PBT.

in this way the number of classification tests, but it reduces also the flexibility of the original PBT to capture a high variability of features.

PBT with Partial Cascading. We observed that a high number of samples can be classified as false by executing only one boosted classifier (see section 5.2). This allows to combine the speed-up of cascading with the flexibilty of the PBT by placing one cascaded classifier in front of the PBT: Our tree model contains one cascading node at the root level. A negative outcome stops the classification immediately, a positive outcome is further processed using the full PBT (figure 1 right).

Classifier Sorting. We also observed that a high amount of samples can be early terminated with a cheap and fast performing classifier (see section 5.2) and that it is advantageous to use expensive classifiers only in places which are executed less often. In our model the most visited place is the cascading node at the root of the tree which can discard a large amount of samples as false. The rest of the tree is visited less frequently. Hence, we sort the expensive classifiers into the later tree nodes while the first node can only use fast executing classifiers.

Image Features. The classifier decides on a per voxel basis if the current voxel belongs to the searched structure or not. Since PBT is a so called ensemble classifier, basically every possible classification method can be integrated. However, the selection of image features has influence on detection performance and execution speed.

In the current work we integrated classifiers which make decisions based on five different image features.

1. Haar-like features with different patterns and sizes.
2. Image intensity
3. Gradients and principal curvatures
4. Region histograms based on image intensity and derivatives with different sampling resolutions and sizes.
5. Structure tensors

Haar-like features and image intensities are the features with the lowest computational costs and are therefore used for building the cascaded root. Gradients need to be computed by filtering as well as principal curvatures which need an additional Hessian analysis step. Region histogram classification multiplies the cost by the number of samples. Structure Tensors require the convolution of the gradient image with a Gaussian kernel and subsequent eigenanalysis of the structure tensor matrix. These three types of classifiers are exclusively used for the non-cascaded part of our PBT.

The chosen weak classifiers are scale variant which is adequate for our application scenario because we expect anatomical structures to have a specific size (small variations in size should be accepted anyway, larger variations because of age or gender can be covered with different detectors and pre-classification based on patient background data).

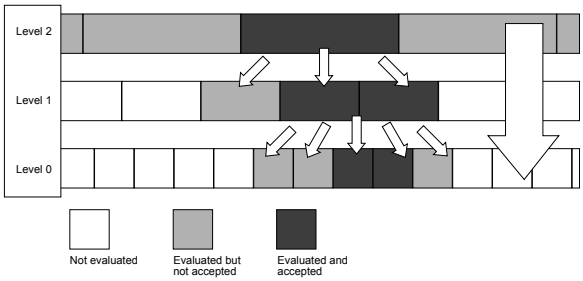


Figure 2: Multiresolution Algorithm

3.2 Multiresolution Feature Detection

The PBT with Partial Cascading is embedded into a multiresolution scheme based on a power of two Gaussian image pyramid [AAB+84] to further reduce the number of voxels to be processed.

A separate classifier C_i is trained for each resolution level. Multiresolution classification starts at the lowest resolution level n by applying classificator C_n on image I_n. Classification results in a set positively marked voxels $(p_0^{+,n},...,p_m^{+,n})$. These voxels are propagated into the next higher resolution level $n-1$ where each positive lower resolution voxel marks the voxels within the corresponding filter kernel in level $n-1$ as candidates. Classification of the current level is only computed on the remaining candidate voxel. The propagation is repeated until the original resolution (level 0) is reached. Figure 2 depicts the algorithm with a 1D example. Note that most of the high resolution voxels do not need to be checked using this scheme.

In the case of overlapping kernels some higher resolution voxels have two or more parent voxels and it can happen that a voxel is marked as positive and negative. In this case the positive mark is kept. This leads to a slight over-segmentation, but on the other hand the effect of false negative samples might be reduced, which is a wanted effect.

3.3 Filtering of Results Using Fast Probability Computation

The direct result of our feature detection algorithm is a bit mask of candidates which still might contain false positives. One method to reduce the number of false positives is to assign a probability to each candidate that reflects the confidence in its classification. The resulting probability map can then be further processed by thresholding which effectively removes outliers and/or non-maximum-suppression which only leaves the candidates which are at the center of the expected shape.

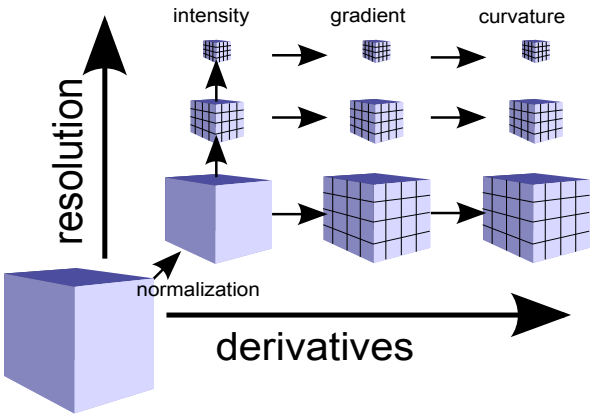


Figure 3: Datastructure: Only the base intensity volume is kept completely in memory. All other data, derivative and lower resolution volumes are computed block wise on demand.

Probabilistic boosting trees can deliver such a probabilistic classification. Drawback of this straight forward approach is the low time performance.

We observed that the result of feature detection form clusters at the feature location resembling already the searched structure (e.g. the intervertebral discs in figure 9). Thus, we propose to assign probabilities to candidates by comparing the shape of its surrounding cluster with the searched shape.

A fast option to compute this are shape particle filters which are applied in our framework. The likelihood that a candidate belongs to the searched structure is computed by applying a shape approximating the structure of interest around each candidate and by measuring the ratio of overlap of neighborhood and shape.

4 IMPLEMENTATION

4.1 Data Preprocessing

The spatial resolution of medical 3D images in a clinical environment is generally highly anisotropic. Especially the slice distances show high variability from modality to modality, from scanner to scanner depending on the used imaging protocol. The scale variant nature of the image features described in section 3.1 requires the same spatial resolution of all images to be processed.

Thus, training data as well as unseen data is preprocessed by resampling the original volume data to an isotropic voxel size that is selected based on the targeted anatomical landmark. The current implementation uses bilinear interpolation for resampling.

The resampled volume (in the following denoted as "base volume") is the basis for all following computations and the original data can be discarded at this point.

4.2 Data Management and Derivative Computation

The data management component is responsible for efficiently providing the necessary data to compute the requested weak classifiers on all resolution levels while keeping the memory footprint small and flexible.

The supported weak classifiers require intensity, gradient and principal curvature data for all positively marked voxel positions on the different levels of resolution. It is obvious that the performance of the weak classifiers decides on the performance of the whole PBT.

It is well known that filtering volume data with separated filters for derivative computation is much faster than applying a three dimensional filters per voxel individually. We currently use a $3 \times 3 \times 3$ Sobel for gradient computation, which can be replaced by any other appropriate separable filter. However, applying a separable filter for derivative computation requires to keep the whole filtered volume in memory, which might be problematic having our initial requirements in mind.

To overcome this limitation and to make the memory footprint manageable also in an environment with limited resources, we introduce a cached block structure (see Figure 3). The intensity base volume is entirely located in memory. Lower resolution volumes, gradients, structure tensors and principal curvature are organized into smaller blocks that are only computed on request. After computation, block data remains cached in memory. If the memory for allocation of new blocks gets low, the cache is partially cleaned by removing data which was accessed the longest time ago.

For fast computation of Haar-like features an additional data structure, an integral volume, is needed. This data is currently computed as a whole and kept in memory. This is due to the more complicated generation method of this data which makes it hard to compute the value block-wise on demand.

4.3 Optimized Classifier Execution

Generally each voxel can be classified individually by executing the whole boosting tree starting from the lowest resolution. However, having in mind that one voxel in a lower resolution volume has influence on a number of voxels in the higher resolution and that the data is arranged in a cached block structure, it is worth to consider a optimal execution order.

Detection of features on the whole volume or of a sub volume follows two strategies. First, feature detection is done in resolution level order. This means that the PBT for one level is executed on the whole region of interest and then all positive classified voxels are propagated to the next higher level.

Second, all per level classification is performed block wise. In this way only a small number of data blocks

must be in cache. Any other execution order (for example line wise) would cause a lot of cache misses and would likely lead to often re-computation of block data. If multiple classifiers must be applied on the same volume all classifiers are executed on each block sequentially. After the first classifier is executed the block cache remains in (partially) filled state. Data which is already cached must not be computed if the next classifier tries to access this data. Parallelization is implemented using a worker thread-pool. Classification of one block is fed into a job queue which distributes the work to the worker threads.

5 EXPERIMENTS

Our multiresolution PBT framework was tested in a real world scenario as preprocessing part for a semi-automatic annotation algorithm for the vertebral column. The task was to preselect appropriate candidates for the location of the intervertebral discs and the spinal canal.

For the intervertebral discs, three different detectors were trained to cover the different appearance of lumbar, thoracic, and cervical disks. The spinal canal could be detected by using only one detector.

5.1 Setup and Training

The algorithm has been trained and evaluated on 19 CT datasets (13 for training 6 for evaluation only) containing different parts of the vertebral column. The datasets have up to 1112 axial slices with a slice resolution of 512×512 and a slice distance between 0.62 *mm* and 3.0 *mm*. Some of the data contains pathologies (broken vertebrae, collapsed disc, scoliotic spines) as well as one cervical dataset from a child.

Experiments have shown that the thinnest intervertebral discs in the cervical section can still be distinguished if the slice distance is at least 1.5 *mm*. We therefore fixed the base volume voxel scale for this experiment as 1.5 *mm* isotropic and the datasets were resampled accordingly.

In all datasets position and location of the intervertebral discs and the spinal column have been manually labeled. Based on the given annotation, positive samples have been generated randomly inside the intervertebral disc and the spinal column. Negative samples have been generated randomly all over the volume with the constraint to have a minimal distance to positive samples of 10 *mm*.

5.2 Performance Evaluation

Time performance of the algorithm has been assessed based on a set of eleven CT volumes (six evaluation and five training datasets). The properties of the data, its original and normalized size is listed in table 1. The classifier is trained using one cascading step and allow only intensity and Haar-like features in the cascade

Volume	original size	normalized size
1	$512 \times 512 \times 202$	$106 \times 106 \times 134$
2	$512 \times 512 \times 163$	$113 \times 113 \times 108$
3	$512 \times 512 \times 361$	$144 \times 144 \times 168$
4	$512 \times 512 \times 222$	$89 \times 89 \times 148$
5	$512 \times 512 \times 249$	$170 \times 170 \times 166$
6	$512 \times 512 \times 152$	$91 \times 91 \times 101$
7	$512 \times 512 \times 277$	$245 \times 245 \times 184$
8	$512 \times 512 \times 260$	$244 \times 244 \times 179$
9	$512 \times 512 \times 1112$	$274 \times 274 \times 370$
10	$512 \times 512 \times 228$	$176 \times 176 \times 228$
11	$512 \times 512 \times 945$	$244 \times 244 \times 630$

Table 1: Properties of volumes for performance evaluation.

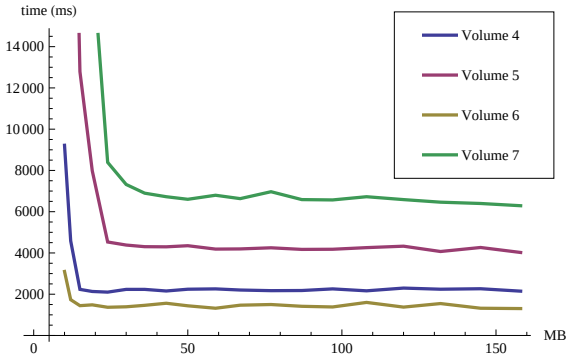


Figure 4: Memory Limits

node. Influence of the different optimizations is measured against this default. Detection performance was measured based on 8 datasets containing the 6 evaluation datasets.

Limited Memory The data structure is designed to cope with limited resources. However, reaching the bounds of memory provokes clearance of cache blocks that might have to be recomputed at a later stage of the algorithm. Figure 4 illustrates the time performance over different cache memory bounds for datasets $4-7$ and show a clear threshold ($\sim$ 25 MB) for all four datasets where the performance/memory ratio changes dramatically. This memory limit is slightly different for each dataset and depends on the dataset size. If the available memory falls below that threshold computation time rises heavily whereas performance remains stable if enough memory is available. The threshold marks the point where data blocks need to be frequently recomputed. As long as enough memory is available deletion of block data from the cache and occasional recomputation has almost no influence on performance.

Multithreading. We tested the multithreading performance of our algorithm on an Intel quad core CPU with 2.4 Ghz and hyperthreading. Figure 5 plots the computation speed over the number of threads again on datasets $4-7$. Time drops until 4 threads are used. For more threads no significant speed-up (but also no significant slowdown) can be monitored.

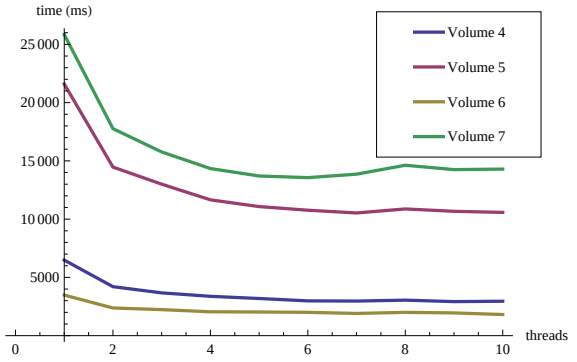


Figure 5: Plot computation time against number of threads. Tested on a quad-core with Hyperthreading.

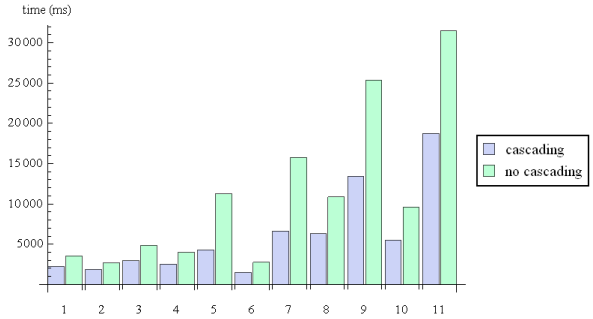


Figure 6: Detection speed comparison between PBT with (blue) and without (green) cascading on eleven different datasets.

The scaling with the number of threads below four is not linear. This is caused by the current locking strategy that prohibits accessing one block if it is currently computed by another thread. This situation mainly occurs if 2nd derivatives have to be computed that require accessing also neighboring first derivative blocks. If another thread is classifying one of these neighboring blocks at the same time it has to wait until the lock is released. This kind of collision happens more frequently as more threads are used. We expect therefore a logarithmic scaling of time performance with the number of cores as long as the locking behavior is not improved.

Cascading Speed-up. The impact of cascading on detection speed has been measured by comparing the time performance of our default detectors with detectors which are trained without including a cascading step. The result is plotted in figure 6. Over eleven datasets we measured a speed-up of $1.45 - 2.67$ for detectors including a cascading step.

Classifier Sorting Speed-up. The impact of classifier sorting is plotted in figure 7. We compare the time performance of our default detector including cascading and sorting with detectors which are allowed to use all classifiers in the cascading node. Classifier sorting results in a speed-up up to 1.65 for detectors which use sorting.

Multiresolution Speed-up. To measure the impact of multiresolution feature detection we compared de-

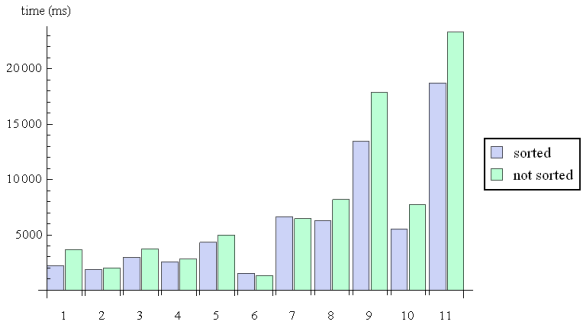


Figure 7: Detection speed comparison between PBT with (blue) and without (green) classifier sorting on eleven different datasets.

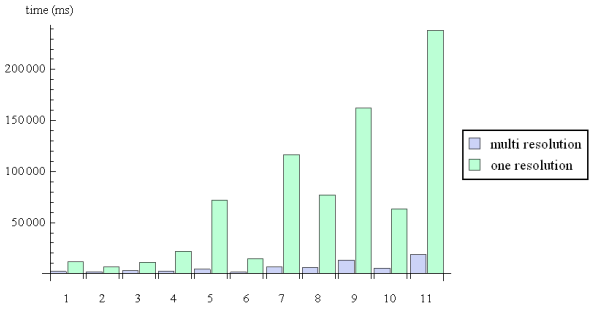


Figure 8: Detection speed comparison between multiresolution vs. one resolution.

tectors using three levels of resolution against detectors using only one level. The results are plotted in figure 8. The measured speed-up ranges between 3.44 and 17.51.

Detection Performance. Two feature detection results are depicted in Figure 9. The first row shows the detection of intervertebral discs in the lumbar section of the spine, the second row the detection of the spinal canal on a whole spine. The detection progress from lowest to highest resolution level is depicted from left to right.

The images illustrate well the effectiveness of the multiresolution scheme since already at the lowest resolution level the major part of the volume is excluded from higher resolution analysis.

The selected voxels (blue) reproduce the shape of the searched anatomical parts to a large extend. However outliers can be observed, for example inside the vertebral body (first row) or at the ventral side of the ribcage (second row). Moreover missing features can be observed as well (first row, ventral side of the topmost disc).

This observation is also reflected in recall and 1-precision plots (figure 10). Recall denotes the ratio between selected voxels within the ground truth and all possible ground truth voxels. 1-precision stays for selected voxels outside the ground truth divided by all the voxels which were selected by the feature detection (also the falsely selected ones). The evaluated data involves healthy spines (1, 2, 3, 6, 7 in figure 10) and

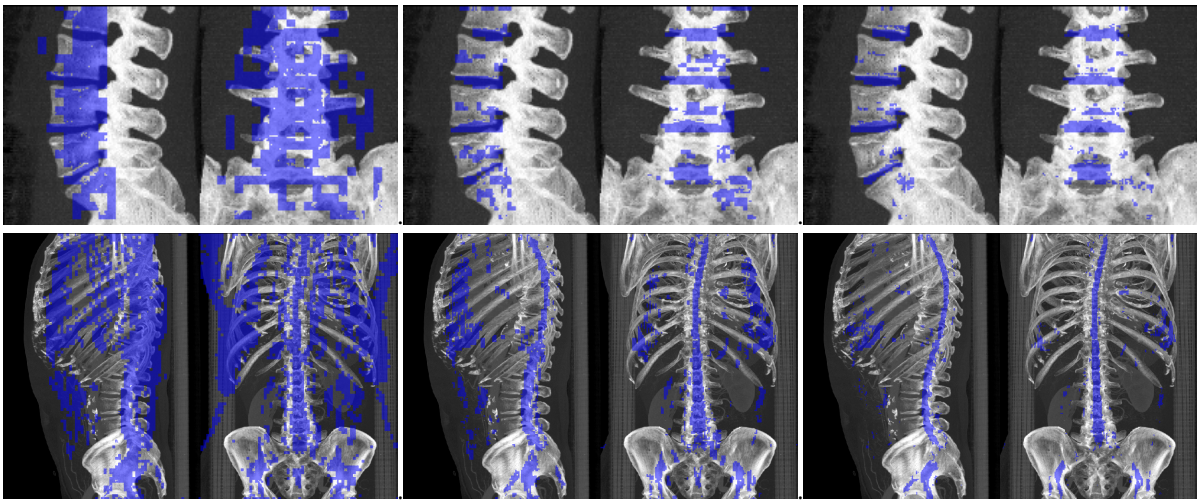


Figure 9: Coronal and sagittal images of detection results for the intervertebral disc (first row) and the spinal column (second row). Three levels of resolution document the detection process, lowest resolution left to highest resolution right.

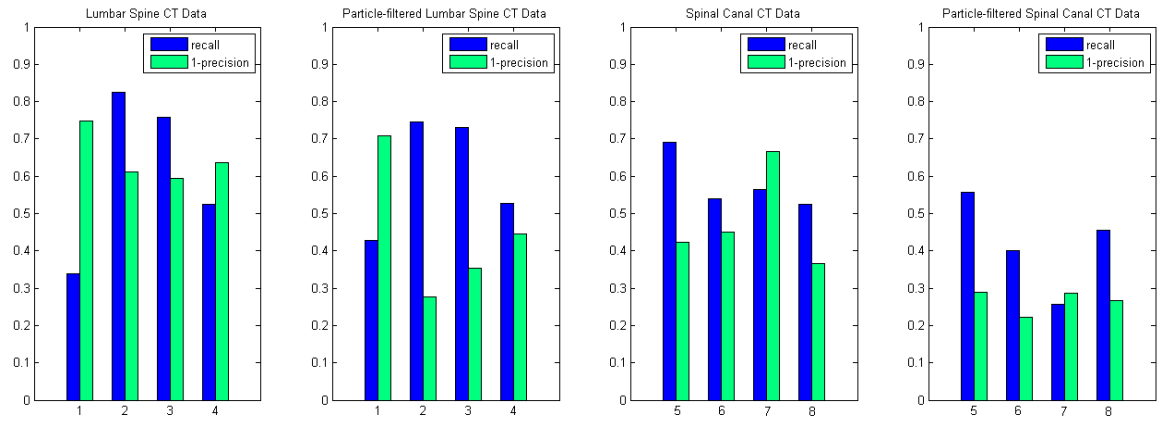


Figure 10: Recall and 1-Precision Plots

spines with diseases like scoliosis and broken vertebrae (4, 5, 8 in figure 10).

The first and the third graph show results after feature detection without any postprocessing where the high recall rates give information about good detection results of structures of interest (discs and spinal canal). However, besides the high recall rates there are also high rates of 1-precisions because of the occurrence of outliers (i.e. spongy bone within vertebrae with similar features to discs). The high 1-precision rates can be reduced by postprocessing steps such as particle filters which are visible in the second and fourth graph of figure 10. The recall rates remain fairly the same, minor reductions are due to moving towards the center voxels of the discs by particle filtering.

An example for postprocessing of the resulting feature mask is depicted in figure 11. First probabilities are computed by applying a box shape particle filter with the dimensions $9 \times 9 \times 60mm^3$. The box approximates elongated shape of the spinal canal. Second, the feature

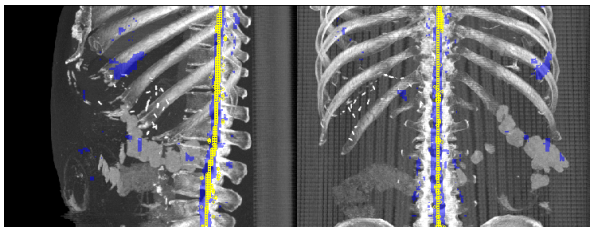


Figure 11: Feature mask (blue) post processed with particle filtering, non-maximum suppression and thresholding (yellow).

points are reduced by non-maximum suppression of the probabilities. Third, outliers are removed by thresholding the probability. The threshold is defined at $t = 0.15$.

6 DISCUSSION AND CONCLUSION

We have presented a method for time and memory efficient feature detection on medical 3D volume data. The goals and requirements formulated at the end of

Section 1 have been reached by selecting a classification based approach based on a Probabilistic Boosting Tree classifier. The classification method was improved by combining the decision tree with one cascading step and the introduction of classifier sorting. This classifier was embedded into a multiresolution framework. We could show that all optimizations together result in a huge time performance gain with an approximated speed-up factor of 20.

Multithread performance was measured to scale non linear (almost logarithmic) which is due to internal data locking. The speed-up is for state of the art quad core CPUs still significant. But to benefit from more parallelism, improvements have to be done in this section. However it is likely that more sophisticated access patterns and locking schemes can help to overcome this problem.

The behavior of the block cache data structure was evaluated in section 5.2. It is noticeable that even larger datasets require only $\sim 25MB$ for the block cache to run almost unhindered. However even under circumstances where less memory is available the algorithm will just perform slower.

Detection rate of this feature detector is not as good as it could be. We believe that other image features and filtering techniques, a finer bases scale and also a different kind of classifier could result in better detection performance. However, trading detection performance against execution speed was a conscious design decision. The results are good enough to use this method to reduce the search space for more specialized and more expensive image processing methods.

REFERENCES

[AAB+84] E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, and J.M. Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[GWGS02] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *Visualization, 2002. VIS 2002. IEEE*, pages 53–60. IEEE, 2002.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[LHJ99] E. LaMar, B. Hamann, and K.I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 355–361. IEEE Computer Society Press, 1999.

[LK08] M. Langer and K.-D. Kuhnert. A new hierarchical approach in robust real-time image feature detection and matching. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1 –4, dec. 2008.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *iccv*, page 1150. Published by the IEEE Computer Society, 1999.

[MDUA07] Christophe Marsala, Marcin Detyniecki, Nicolas Usunier, and Massih-Reza Amini. High-level feature detection with forests of fuzzy decision trees combined with the rankboost algorithm. Technical report, Université Pierre et Marie Curie-Paris, 2007.

[MV09] A. Militzer and F. Vega-Higuera. Probabilistic boosting trees for automatic bone removal from CT angiography images. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7259 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, February 2009.

[NGL+09] M. Niemeijer, M.K. Garvin, K. Lee, B. van Ginneken, M.D. Abràmoff, and M. Sonka. Registration of 3D spectral OCT volumes using 3D SIFT feature point matching. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7259, page 51, 2009.

[Tu05] Z. Tu. Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering. In *ICCV*, volume 2, pages 1589 –1596, 2005.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages I–511 – I–518, 2001.

High-Quality Cartographic Roads on High-Resolution DEMs

Mikael Vaaraniemi
BMW Forschung und Technik GmbH
München, Germany
mikael.va.vaaraniemi@bmw.de

Marc Treib Rüdiger Westermann
Technische Universität München
München, Germany
{treib,westermann}@tum.de

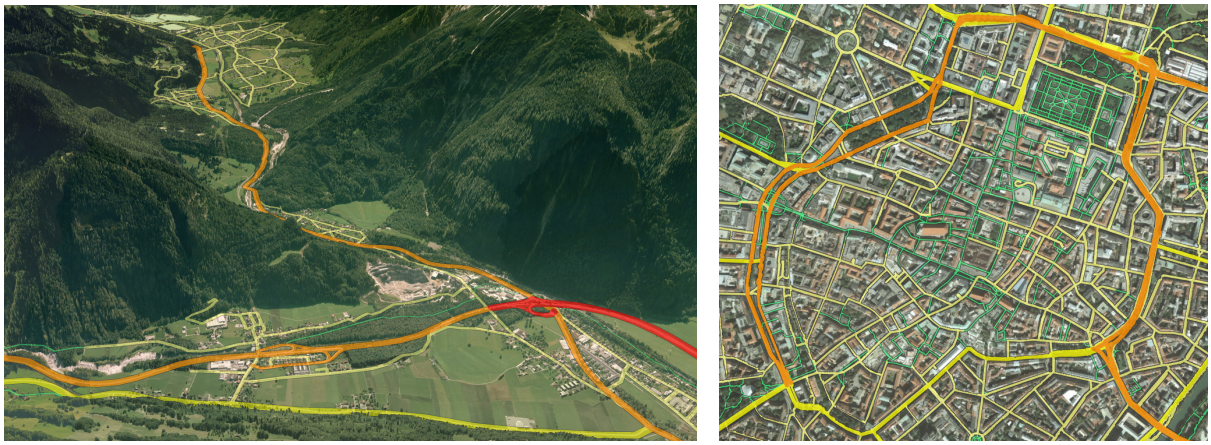


Figure 1: Cartographic rendering of roads in the Vorarlberg region, Austria, and in central Munich, Germany.

ABSTRACT

The efficient and high quality rendering of complex road networks—given as vector data—and high-resolution digital elevation models (DEMs) poses a significant problem in 3D geographic information systems. As in paper maps, a cartographic representation of roads with rounded caps and accentuated clearly distinguishable colors is desirable. On the other hand, advances in the technology of remote sensing have led to an explosion of the size and resolution of DEMs, making the integration of cartographic roads very challenging. In this work we investigate techniques for integrating such roads into a terrain renderer capable of handling high-resolution data sets. We evaluate the suitability of existing methods for draping vector data onto DEMs, and we adapt two methods for the rendering of cartographic roads by adding analytically computed rounded caps at the ends of road segments. We compare both approaches with respect to performance and quality, and we outline application areas in which either approach is preferable.

Keywords

cartography, vector draping, shadow volume, GIS, roads, terrain.

1. INTRODUCTION

Geographic Information Systems (GIS) store, analyze and visualize geo-referenced data. Road networks, land usage regions and selected points of interest are usually stored as vector data. In urban planning, cartography, and for navigation purposes, the visualization of roads on digital terrain models plays an important role [Döl05]. GIS engines should be able to handle and display such vector data efficiently and at high quality. A bare and uncluttered visualization as in paper maps is desirable. This *cartographic* representation of roads requires vivid colors, dark edges, rounded caps and runtime scaling of road width [Kra01, RMM95]. Dynamic scaling allows the perception of roads at every distance. In a cartographic rendering, roads are tinted using vivid colors to distinguish them from the underlying terrain. Associating different colors to each type of road induces an automatic cognitive grouping of similar roads [Kra01]. In addition, dark edges around roads add visual contrast [RMM95]. Examples of such cartographic representations are shown in Fig. 1 and 2. An additional aspect of a cartographic representation are rounded caps at each road segment. This avoids the ap-

Figure 2: Cartographic rendering of road maps using vivid colors and dark edges to achieve a high visual contrast to the underlying terrain.

pearance of cracks between segments and makes the visualization more appealing by introducing smooth endings and avoiding undesirable angular corners.

Another important information layer in GIS is the digital elevation model (DEM). It is usually given as raster data defining a 2.5D height map. Since the resolution and size of these DEMs are increasing rapidly, rendering approaches must be capable of dealing with TBs of data and gigantic sets of primitives that have to be displayed at high frame rates. To cope with these requirements, visualization techniques employ adaptive Level-Of-Detail (LOD) surface triangulations [LKR+96] and data compression, combined with sophisticated streaming and pre-fetching strategies [DSW09]. In such scenarios, the combined visualization of roads and a high-resolution DEM in a single visualization engine becomes a challenging task.

The main contribution of this paper is a method for rendering cartographic roads with rounded caps on high-resolution DEMs. We extend existing vector draping methods by introducing the possibility to compute caps analytically, thus avoiding an explicit triangulation. In this way we achieve a high-quality appearance without increasing the number of geometric primitives to be rendered. Furthermore, we introduce screen-space road outlines, runtime width scaling, and correct treatment of road intersections.

We have integrated our method into a tile-based terrain rendering engine. During preprocessing, this engine builds an multiresolution pyramid for both the DEM and the photo texture. It then partitions each level into square tiles, creating a quad tree. Each tile stores a Triangulated Irregular Network (TIN) representation of the DEM along with the photo texture. During runtime, tiles are chosen based on a maximum allowed screen-space error. In combination, this enables interactive 3D browsing of high-resolution terrain data with superimposed cartographic roads.

2. RELATED WORK

Terrain Rendering. Terrain rendering approaches using rasterization have been studied extensively over the last years. They employ the GPU to render large sets of polygonal primitives, and they differ mainly in the hierarchical height field representation used. There is a vast body of literature related to this field and a comprehensive review is beyond the scope of this paper. However, Pajarola and Gobbetti [PG07] discuss the basic principles underlying such techniques and provide many useful algorithmic and implementation-specific details. A thorough discussion of terrain rendering issues that are specifically related to high resolution fields is given in [DSW09].

Vector Data. The mapping of vector data on DEMs is an active research subject. The existing methods can be broadly classified into geometry-based, texture-based and shadow volume-based approaches.

Geometry-based methods generate and render separate primitives for the vector data. As the sampling frequency of the vector data generally does not match the triangulation of the underlying terrain, an adaption to the terrain triangulation and its LOD scheme is necessary. Because of this preprocess, geometry-based algorithms are strongly tied to the terrain rendering system and usually only allow static vector data [ARJ06, SGK05, WKW+03].

Texture-based techniques map the vector data onto a DEM in two steps: first, the data is rendered into off-screen textures either at runtime or in a preprocess. Afterwards, these textures are overlaid onto the terrain using texture mapping [DBH00]. This approach does not produce any aliasing artifacts thanks to hardware-supported texture filtering. Additionally, these methods are independent of the underlying terrain triangulation algorithms.
Static texturing methods provide high performance, but do not allow runtime changes of rendering parameters. Further problems occur at large zoom factors, as only limited resolution textures can be precomputed—there is an inherent tradeoff between the memory requirements and the obtainable quality [DBH00]. Therefore, Kersting and Döllner [KD02] combine this approach with on-demand texture pyramids: associating each quadtree region with an equally sized texture allows on-the-fly generation of appropriate textures. Dynamic vector data can be visualized if these textures

are created in each frame. However, this severely impacts performance, as many render target switches are needed. To overcome this, Schneider et al. [SGK05] introduce an approach using a single reparameterized texture for the vector data, analogously to perspective shadow mapping (PSM) [SD02]. As in PSM, some aliasing artifacts occur.

Bruneton and Neyret [BN08] present an approach that adapts the terrain heightfield to constraints imposed by the vector data (e.g. to enforce locally planar roads). Their method works only on regular meshes and would be difficult to generalize to our TIN-based terrain system. It is also not feasible for high-resolution terrain data. Additionally, an adaption of the heightfield is only necessary if the terrain resolution is insufficient to resolve such constraints, or if real-time editing is desired.

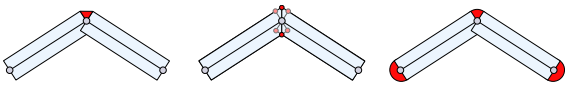
A **shadow volume-based** approach, recently introduced by Schneider and Klein [SK07], uses the stencil shadow volume algorithm to create pixel-exact draping of vector data onto terrain models. A stencil mask is created by extruding polygons along the nadir and computing the screen-space intersection between the created polyhedra and the terrain geometry. Using this mask, a single colored fullscreen quad is drawn. For each color, a separate stencil mask has to be generated. However, as the number of different vector data colors is typically small, this is not a major problem. The approach does not require any precomputations and is thus completely independent of the terrain rendering algorithm.

Our goal is to render *cartographic* roads on a *high-resolution* DEM. Continuous road scaling is a prerequisite, which makes texture-based approaches unsuitable. Likewise, a runtime triangulation of roads to match the DEM is not feasible, so most existing geometry-based approaches are not usable in our case.

We chose to use the shadow volume approach, as it does not require a preprocess and thus allows for runtime scaling of roads. It also provides pixel-exact projections. As a simpler and faster alternative, we also investigate a geometry-based approach where we adapt only the road centerlines to the DEM, so road scaling remains possible.

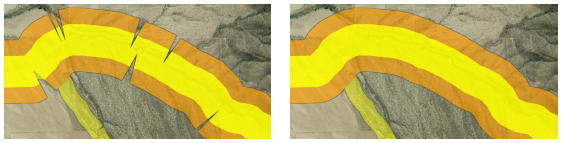
3. CARTOGRAPHIC ROADS

In GIS, roads are usually stored as vector data, i.e. as a collection of 2D polylines. One possibility to visualize such data is to convert the vector data into geometric primitives that are rendered on top of the terrain. However, a naive extrusion of each line segment to a quad results in the appearance of cracks between segments. The higher the curvature of a polyline, the more these cracks become visible. Two pragmatic solutions exist: filling the holes with additional triangles (see Fig. 3(a)) or connecting the corners of adjacent quads (see Fig. 3(b)). Both solutions are only possible if adjacency information is available. In real data sets, however, this information is commonly incomplete. Fig. 4 shows an example from a real data set where one continuous road is represented by several individual polylines, resulting in cracks between adjacent road segments where the polylines meet. We therefore choose a robust and elegant solution, which draws caps to avoid the appearance of cracks (see Fig. 3(c)) and does not require adjacency information. In addition to filling cracks, this approach generates visually pleasant smooth road endings (see Fig. 5, top). It also naturally handles sharp turns in a road (Fig. 5, bottom). Many major navigation systems visualize roads using rounded caps, for example Nokia with Ovi Maps, Google with Google Maps, Navigon and TomTom. It has become a de-facto standard technique when rendering cartographic roads [Phy09]. A naive method for rendering caps is the triangulation of a half-circle, leading to a large number of additional triangles per segment. Furthermore, the discrete triangulation becomes visible at large zoom factors. In the following sections, we present two methods that allow using perfectly round caps while avoiding an increase of the triangle count.



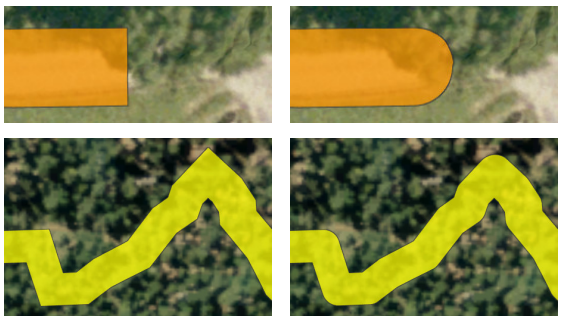
(a) Extra triangle (b) Moving corners (c) Rounded caps

Figure 3: Methods for removing cracks between quads.



(a) Cracks (b) Fixed with caps

Figure 4: Cracks occur because of missing adjacency information.



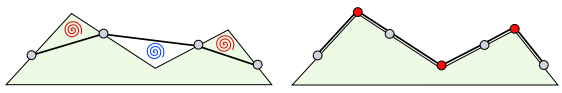
(a) No rounded caps (b) With rounded caps

Figure 5: Quality improvement with rounded caps.

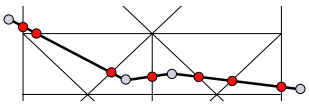
4. GEOMETRIC APPROACH

Our first method renders cartographic roads using a geometry-based approach. From the initial polyline

representation of a road, we individually process each line segment defined by successive vertices. In a preprocess, these lines are clipped against the terrain mesh in 2D, inserting additional vertices at each intersection (see Fig. 6). For more details on this preprocess, see section 6.1.



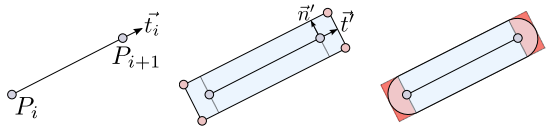
(a) Incorrect mapping of a road (grey vertices); problematic areas are marked by spirals (b) Correct mapping of the road using additional vertices (red)



(c) Top-down view with additional vertices

Figure 6: Geometry-based mapping of roads onto a terrain mesh.

To render rounded caps, we do not explicitly triangulate half-circles at the beginning and the end of each road segment. Instead, we render a single quad encompassing an entire road segment and evaluate the caps analytically in a shader program [Gum03] (see Fig. 7).



(a) Line segment i (b) Generated quad (c) Analytical caps

Figure 7: Analytical evaluation of rounded caps on a base quadrilateral.

We use the endpoints P_i and P_{i+1} of each line segment and the tangent $\vec{t}_i$ to generate a quad encompassing both capped ends (see Fig. 7(a) and (b)).

The caps are cut out of the generated quad in a pixel shader. We create a normalized local coordinate system inside both caps [RBE+06], which allows determining those fragments of a quad that are outside the cap and have to be discarded (see Fig. 7(c)).

Given points P_0, P_1, the ratio h between their distance $d = \overline{P_0P_1}$ and the cap radius $\frac{w}{2}$ is given by

$$h = \frac{w}{(d + 2 \cdot \frac{w}{2})} = \frac{w}{d+w}.$$

Equipped with h, we generate the local coordinates inside the caps with

$$x_{cap} = \frac{|x|-1}{h} + 1 \quad , \qquad y_{cap} = |y|.$$

If $x_{cap} > 0$, the fragment lies inside the cap area (the red area in Fig. 7(c)). If $x_{cap}^2 + y_{cap}^2 > 1.0$, it is outside of the half circle that builds the cap, and is discarded.

5. SHADOW VOLUME APPROACH

Our second algorithm is an extension of the shadow volume-based approach introduced by Schneider and Klein [SK07]. We extrude the road geometry along the nadir and apply a stencil shadow volume algorithm [Cro77, Hei91]. Thus, we compute the intersections between the extruded roads with the terrain geometry, resulting in per-pixel accurate projections onto the terrain. Similar to the approach described in section 4, we extend this algorithm by adding analytic rounded caps. We enlarge the geometry of each line segment to encompass the caps, and construct a local coordinate system that allows us to determine the fragments lying inside or outside the cap area. In the inside area, we analytically evaluate the caps via an intersection test between a ray and a cylinder and compute the depth value of the intersection point to be used during the depth test.

5.1. Intersection

From the camera position O, the fragment position F, and the view direction $\vec{v} = (F-O)/|F-O|$ we construct the view ray $R = O + t\vec{v}$. Given such a ray, the intersection of the ray with the cylinder spanned by the cap can be computed. Because the cylinder is always aligned with the z axis (the nadir), we can replace the 3D ray-cylinder test by a 2D ray-circle test in the xy plane (see Fig. 8).

A circle with center C and radius r is defined by the equation

$$(X-C)^2 = r^2.$$

Inserting the ray R into this equation with $\vec{c} := O - C$ yields

$$((O + t\vec{v}) - C)^2 = (\vec{c} + t\vec{v})^2 = r^2.$$

Expanding this results in the quadratic equation

$$(\vec{v}\cdot\vec{v})\, t^2 + 2\,(\vec{v}\cdot\vec{c})\, t + (\vec{c}\cdot\vec{c} - r^2) = 0.$$

Solving for t gives the discriminant

$$d = 4\,(\vec{v}\cdot\vec{c})^2 - 4\,(\vec{v}\cdot\vec{v})\,(\vec{c}\cdot\vec{c} - r^2).$$

If $d \leq 0$, there is none or only a single solution to the quadratic equation. This means that the ray does not hit the cap at all, or just grazes it. In this case, we discard the fragment. Otherwise, we get

$$t_{1/2} = \frac{-2\,(\vec{v}\cdot\vec{c}) \;\pm\; \sqrt{d}}{2\,(\vec{v}\cdot\vec{v})}.$$

For front faces, $\min(t_1, t_2)$ is the correct solution, for back faces it is $\max(t_1, t_2)$.

So far, we have assumed that the road geometry is extruded toward infinity to generate the shadow volumes. Since this is wasteful in terms of rasterization fill rate, we consider the height field for limiting the extent of

the shadow volumes. Assuming the terrain being partitioned into tiles, it is sufficient to extrude each line segment only within the extent of the bounding box of the tile it belongs to.
To accommodate this, the intersection algorithm has to be extended to handle the top and bottom sides of the extruded polyhedron: If the 2D distance between F and C is smaller than the cap radius (which can only happen for fragments belonging to the top or bottom side), F already gives the final intersection.

5.2. Numerical Precision

The algorithm as presented so far suffers from problems caused by limited numerical precision. One such problematic situation is depicted in Fig. 8: The intersection between each ray and the cylinder is computed twice, once for the front face of the bounding box (corresponding to F_0 in the figure) and once for the back face (corresponding to F_1). The ray direction is computed as $F_0 - O$ and $F_1 - O$, respectively. Because of small perturbations in F_0 and F_1, which are caused by the limited precision of the interpolation hardware, one of the intersection tests may report an intersection while the other one does not. This results in inconsistent output causing visible artifacts.

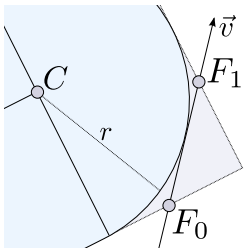


Figure 8: Numerically problematic ray-circle intersection.

In order to achieve consistent results, we compute both intersections in the same shader invocation: We render the geometry with front face culling enabled, and analytically compute the entry point into the bounding box of the extruded road. We then compute both intersections between the ray and the road as described above. This results in two depth values z_0, z_1 that need to be compared to the terrain depth z_t. We therefore replace the regular depth test with a custom two-sided test: z_t is read from a texture created as a secondary render target during the terrain rendering pass. If $z_0 < z_t < z_1$, then the road volume intersects the terrain geometry; otherwise, we discard the fragment.

Two beneficial side effects of this approach are that only half the amount of geometry needs to be rasterized compared to the naive approach, and that in contrast to the original shadow volume algorithm it does not require the rendering of full-screen quads to color the intersections.

6. IMPLEMENTATION DETAILS

In our proposed GIS engine, we visualize vector data e.g. from the OpenStreetMap project [Ope10]. Road networks are stored as a collection of polylines. Each polyline has a *functional road class* (FRC) [Tal96], defining a distinct width and color. For efficient data management at runtime, we partition the vector data into quadtree tiles, similar to the terrain data. Inside each tile, roads are stored sorted by their FRC.

6.1. Geometry Clipping

To avoid an incorrect mapping of roads onto the DEM in the geometric approach as in Fig. 6(a), we apply a preprocess where the centerline of each road segment is clipped against the terrain mesh in 2D. Additional vertices are inserted at each intersection (see Fig. 6(c)). However, finding the exit point of a line in a triangle by line-line intersection tests with the triangle edges provides poor numerical stability. We therefore perform these calculations in barycentric coordinates as illustrated in Fig. 9.

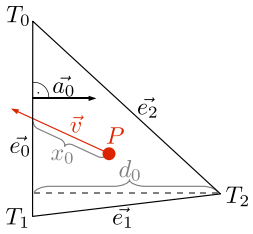


Figure 9: Computing line – triangle edge intersections.

We trace a line starting at point P along the normalized direction vector $\vec{v}$ in the triangle defined by the vertices T_0, T_1 and T_2. The change in the barycentric coordinate λ_2 of P with respect to T_2 is given by the signed distance moved along $\vec{a_0}$ divided by the distance d_0 of T_2 from $\vec{e_0}$, where $\vec{a_0}$ is a normalized vector perpendicular to $\vec{e_0}$ and pointing inside the triangle. When moving along $\vec{v}$, this becomes $(\vec{a_0} \cdot \vec{v})/d_0$. If this value is larger than zero, $\vec{v}$ is pointing away from $\vec{e_0}$ and we skip this edge. Otherwise, the maximum distance x_0 we can move along $\vec{v}$ before we hit $\vec{e_0}$ is given by

$$x_0 = \frac{\lambda_2 d_0}{\vec{a_0} \cdot \vec{v}}.$$

This can be done analogously for the other edges to compute x_1 and x_2; the smallest of these provides the actual exit point. At this point, an additional vertex is inserted into the polyline.

6.2. Cartographic Rendering

Scaling. In cartographic rendering, roads should be visible at all zoom levels. Therefore, while zooming out our system scales the roads' widths continuously.

The scaling factor is determined by the distance to the viewer. To avoid that roads close to the viewer become too wide, we only scale roads that are further away from the user than a given distance threshold (see Fig. 10).

(a) No scaling (b) Constant scaling (c) Distance-based scaling

Figure 10: Scaling of road width. Without scaling, distant roads become too narrow (left). A constant scale makes close roads too wide (middle). Distance-based width scaling gives satisfactory results (right).

Intersections. At crossroads or junctions, multiple roads of potentially different FRCs overlap, resulting in visible artifacts caused by additive blending. To resolve this problem, we draw roads into an offscreen render target without blending, in increasing order of importance.
The same approach allows for an easy integration of multi-colored roads by drawing a road multiple times with different widths and colors. This increases the geometry count proportionally to the number of colors, but since typically only a few important roads use multiple colors, this is acceptable. Fig. 11 demonstrates the correct handling of intersections of roads with different FRCs, including a two-color motorway.

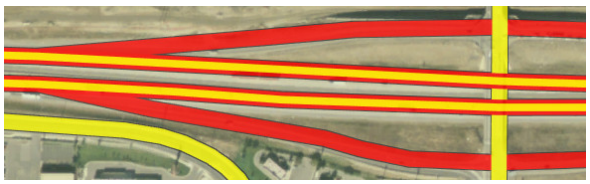


Figure 11: Correct handling of road intersections.

Outlines. To distinguish cartographic roads from the underlying terrain, we add dark edges around roads to increase contrast [RMM95]. To detect edges in screen space, we use a 3×3 or 5×5 kernel to find the local maximum road intensity $\alpha_{\max}$ around each fragment. The road intensity is the road opacity for pixels which are covered by a road, and 0 otherwise. The difference $\alpha_{\max} - \alpha_{\text{current}}$ defines the resulting edge intensity. Fig. 12 demonstrates the increase in visibility achieved by using outlines around roads.

(a) Without outlines (b) With outlines

Figure 12: Improving visibility by using dark outlines.

7. RESULTS

We have tested the proposed algorithms using three high-resolution data sets:

- A DEM of the US State of Utah, covering an area of about $276,000$ km^2 at a geometric resolution of 5 m. The road data set contains about $6,839,000$ vertices (216 MB).
- A DEM of Bavaria in Germany, covering an area of about $70,500$ km^2 at a geometric resolution of up to 80 cm. The road data set contains about $5,697,000$ vertices (151 MB).
- A DEM of the Vorarlberg region in Austria, covering an area of about $4,760$ km^2 at a geometric resolution of 1 m. The road data set contains about $213,000$ vertices (7 MB).

The size of the terrain data including photo textures is around 1 TB per data set. We therefore use an out-of-core visualization system capable of handling arbitrarily large data sets.
The preprocessing step for the geometric approach (see section 6.1) increased the size of the road data by about a factor of ten in all tested cases. Note that for the shadow volume approach, this step is not required.

Performance. All performance measurements were taken at a display resolution of 1600×1200 on a PC with Windows Vista, a 2.66 GHz Intel Core 2 Quad CPU, 8 GB of RAM and an ATI Radeon HD 5870 GPU (driver version 10.6).
The graph in Fig. 13 shows the frame rate during a recorded flight over the medium-resolution DEM of Utah at an average speed of about 1750 m/s. When rendering geometric roads (GEO), the maximum (average) performance drop is about 30% (26%) compared to rendering the terrain without roads. The highest performance impact occurs over Salt Lake City (far right in the graph). This area contains a dense road network and only a small amount of terrain geometry, as buildings are not included in the height field. The additional rendering of rounded caps does not significantly influence the performance.
For shadow volume-based roads (SV), the maximum (average) performance drop is around 40% (35%) without and 55% (42%) with rounded caps. Breaking the numbers down to the sole rendering of roads, SV with caps is about 1.4 times as expensive as without caps.

The visual quality produced by both techniques is identical at most locations in Utah. Therefore, GEO is preferable because of its higher performance. Fig. 14

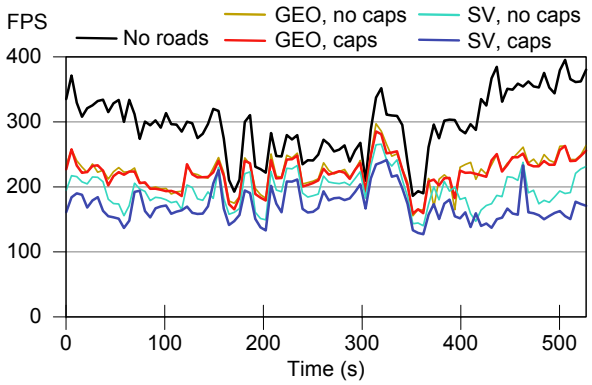


Figure 13: Performance - Utah

shows the frame rates during a flight over the high-resolution data set of Bavaria at an average speed of about 950 m/s. In this scenario, the performance of all approaches is very close; the average cost is between 33% and 43%. Even though GEO often requires many more triangles (up to about 3 million) than SV ($\leq$ 1M) because of the adaption to the terrain mesh (which itself uses up to about 35M triangles), GEO is still slightly faster. Thus, the GPU is more limited by shading computations than by the geometry throughput. However, GEO can often not provide an adequate mapping on such high-resolution terrain data (see Fig. 15). Therefore, SV is preferable for such fine-grained DEMs.

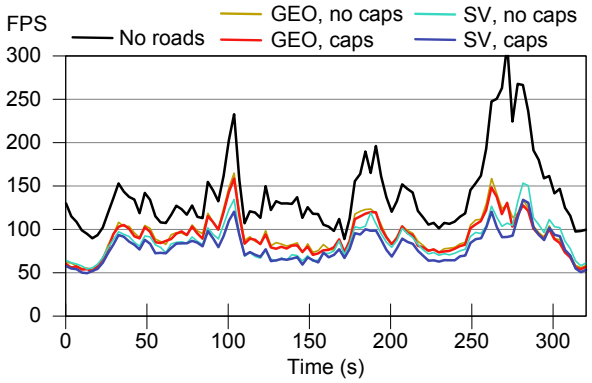
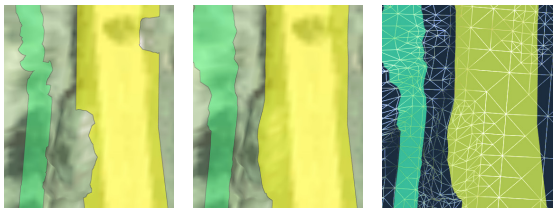


Figure 14: Performance - Bavaria

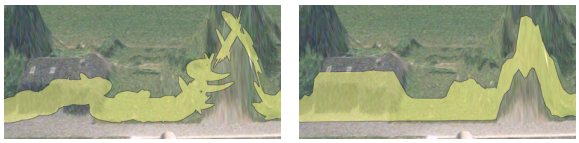


(a) Geometric (b) Shadow volume (c) Wire frame

Figure 15: Comparison of our draping algorithms on high-resolution terrain.

The Vorarlberg data set has a similar geometric resolution as the Bavaria data set, but the road network is much more sparse. Regardless of which algorithm is chosen for rendering roads, the highest performance impact amounts to only 15%. However, as in Bavaria, GEO can not provide adequate quality.

Matching. We should note that in many situations the vector data set did not exactly match the terrain data, i.e. there was a certain offset between the vector data roads and roads in the phototextures. These problems frequently occur in cities or forests, where even a slight offset causes a road to be projected onto a building or a tree. GEO fails to produce any reasonable results in this case (see Fig. 16(a)); SV produces a technically correct but not very useful projection (see Fig. 16(b)). This is a problem of the data rather than the draping algorithm. An additional preprocessing step could match the vector data to the terrain and its phototextures.



(a) Geometric (b) Shadow volume

Figure 16: Artifacts caused by a mismatch between terrain and vector data.

Comparison. Our method presents a marked improvement over several commercial GIS systems. For example, Google Earth 6.0 uses a simple geometric approach without adaption to the terrain and therefore does not achieve a correct projection of roads onto the DEM. It also does not provide correct road intersections and does not support multi-color roads or outlines. ArcGIS 10.0 rasterizes vector data into textures which are overlaid onto the terrain, similar to the orthophotos. This results in a correct projection and correct behavior at road intersections. However, a dynamic scaling of road widths is not possible, and multi-color roads or outlines are not supported.

8. CONCLUSION

In this paper, we have proposed and evaluated two approaches for rendering high-quality cartographic roads with rounded caps on high-resolution 3D terrain models. Both can be used on hardware platforms supporting Direct3D 10 or OpenGL 3.0. We have shown that a geometry-based approach provides high performance and good quality for low- to medium-resolution terrain data sets. However, it requires a moderately complex preprocessing step, and it can not provide an adequate visual quality with high-resolution terrain data. It is therefore a reasonable choice for low-end hardware, e.g. on mobile devices, where rendering of high-resolution terrain data is not feasible.

The shadow volume algorithm enables pixel-exact rendering of cartographic roads on 3D terrain. It is more expensive at runtime than the geometry-based approach; however, the rendering of high-resolution terrain remains the larger part. In low-resolution terrain data sets, on the other hand, its relative performance impact is large. The algorithm is easy to integrate into existing terrain rendering engines, as no adaption of roads to the terrain is required. It also extends naturally to polygonal vector data.

In further research, we plan to evaluate the use of tesselation shaders for the creation of geometric caps on Direct3D 11 or OpenGL 4.0 capable hardware.

9. ACKNOWLEDGEMENTS

The authors wish to thank the Landesvermessungsamt Feldkirch, Austria, the Landesamt für Vermessung und Geoinformation Bayern and the State of Utah for providing high-resolution geo data.
This publication is based on work supported by Award No. UK-C0020, made by King Abdullah University of Science and Technology (KAUST).

10. REFERENCES

[ARJ06] Agrawal, A., Radhakrishna, M., and Joshi, R. Geometry-based mapping and rendering of vector data over LOD phototextured 3D terrain models. In *Proceedings of WSCG*, pages 787–804, 2006.

[BN08] Bruneton, E. and Neyret, F. Real-time rendering and editing of vector-based terrains. In *Comput. Graph. Forum*, volume 27, pages 311–320, April 2008. Special Issue: Eurographics '08.

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

[DBH00] Döllner, J., Baumann, K., and Hinrichs, K. Texturing techniques for terrain visualization. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*, Washington, DC, USA, 2000. IEEE Computer Society.

[Döl05] Döllner, J. Geovisualization and real-time 3d computer graphics. In E., Dykes, J., MacEachren, A., and Kraak, M., editors, *Exploring Geovisualization*, chapter 16, pages 325–343. Pergamon, 2005.

[DSW09] Dick, C., Schneider, J., and Westermann, R. Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphics Forum*, 28(1):67–83, 2009.

[Gum03] Gumhold, S. Splatting illuminated ellipsoids with depth correction. In *Proceedings of 8th International Fall Workshop on Vision, Modelling and Visualization*, volume 2003, pages 245–252, 2003.

[Hei91] Heidmann, T. Real shadows, real time. *IRIS Universe*, 18:28–31, 1991.

[KD02] Kersting, O. and Döllner, J. Interactive 3d visualization of vector data in GIS. In *GIS '02: Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 107–112, New York, NY, USA, 2002. ACM.

[Kra01] Kraak, M. *Cartographic principles*. CRC, 2001.

[LKR+96] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. A. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA, 1996. ACM.

[Ope10] OpenStreetMap. OpenStreetMap website, 2010.

[PG07] Pajarola, R. and Gobbetti, E. Survey of semi-regular multiresolution models for interactive terrain rendering. *Vis. Comput.*, 23(8):583–605, 2007.

[Phy09] Physical Storage Format Initiative. *Navigation Data Standard: Compiler Interoperability Specification*, 2009.

[RBE+06] Reina, G., Bidmon, K., Enders, F., Hastreiter, P., and Ertl, T. GPU-Based Hyperstreamlines for Diffusion Tensor Imaging. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2006*, pages 35–42, 2006.

[RMM95] Robinson, A., Morrison, J., and Muehrcke, P. *Elements of cartography*. John Wiley & Sons Inc, 1995.

[SD02] Stamminger, M. and Drettakis, G. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562, New York, NY, USA, 2002. ACM.

[SGK05] Schneider, M., Guthe, M., and Klein, R. Real-time rendering of complex vector data on 3d terrain models. In Thwaites, H., editor, *The 11th International Conference on Virtual Systems and Multimedia (VSMM2005)*, pages 573–582. ARCHAEOLINGUA, October 2005.

[SK07] Schneider, M. and Klein, R. Efficient and accurate rendering of vector data on virtual landscapes. *Journal of WSCG*, 15(1-3), January 2007.

[Tal96] Talvitie, A. *Functional Classification of Roads*. Transportation Research Board, Washington, D.C., 1996.

[WKW+03] Wartell, Z., Kang, E., Wasilewski, T., Ribarsky, W., and Faust, N. Rendering vector data over global, multi-resolution 3d terrain. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 213–222, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

Pipeline Reconstruction from Fisheye Images

Yuhang Zhang
The Australian National University
yuhang.zhang@anu.edu.au

Richard Hartley
The Australian National University
richard.hartley@anu.edu.au

John Mashford
CSIRO
Australia
john.mashford@csiro.au

Lei Wang
The Australian National University
lei.wang@anu.edu.au

Stewart Burn
CSIRO
Australia
stewart.burn@csiro.au

ABSTRACT

Automatic inspection of pipelines has great potential to increase the efficiency and objectivity of pipeline condition assessment. 3-D pipeline reconstruction aims to reveal the deformation of the pipe surface caused by internal or external influences. We present a system which can reconstruct the inner surface of buried pipelines from multiple fisheye images captured inside the pipes. Whereas the pipelines are huge, a fatal defect can be as tiny as a fine crack. Therefore a reliable system demands both efficiency and accuracy. The repetitive patterns on the pipe surface and the poor illumination condition during photographing further increase the difficulty of the reconstruction. We combine several successful methods found in the literature as well as new methods proposed by ourselves. The proposed system can reconstruct pipe surface not only accurately but also quickly. Experiments have been carried out on real pipe images and show promising performance.

Keywords: 3D reconstruction, surface reconstruction, fisheye lens, water pipelines, pipe inspection, image processing.

1 INTRODUCTION

Water pipelines are indispensable facilities of modern urban systems. After serving for decades underground, the condition of the pipelines deteriorates to varying degrees. Timely inspection and repair is therefore required to prevent imminent collapse. Traditionally pipe inspection involves intensive manual effort. Manual image interpretation is an expensive process for which wrong decisions caused by fatigue and subjective bias are inevitable. Hence a computer-aided inspection system is of great value.

We present a system which can reconstruct the inner surface of buried water pipes based on a sequence of images captured inside the pipes (Figure 1). Deformation of the pipe surface which foreshadows the pipeline collapse can then be detected from the reconstructed model. Early work on similar applications relied on range cameras such as laser scanners, which is expensive. Later, due to the developments of computer vision, methods solely based on 2D images were proposed [3, 8, 9]. However, because of the limitation in computer vision at the time and the difficulty in this particular application, some of these works made restrictive assumptions such as that, the pipes are built from bricks which provide distinctive patterns; and the others terminate with reconstructing a small group of isolated points only. Some 3D reconstruction applications of general scenes [18] bear the same limitation as well. More recently, several large-scale 3D reconstruction applications of general scenes have been proposed [1, 20], which can reconstruct millions of points in a relatively short time. However, their implementation requires high-end parallel computers.

What distinguish the proposed system from all the previous ones are:

1. we intensively reconstruct the pipe surface, which is composed of millions of points, rather than a group of selected points;
2. our algorithm is fast and can be implemented on normal PCs;
3. we have proposed a number of specific mechanisms to increase the robustness of the system, so that it can work with pipe surface without distinctive patterns under poor illumination conditions.

We first give an overview of the reconstruction problem, as well as our method. When discussing each step in detail, experimental results will be provided accordingly. As will be seen, our method performs not only accurately but also quickly.

2 OVERVIEW

We make no particular assumption about the material of the pipelines. Actually, the pipes a civil engineer frequently confronts are made from concrete which gives

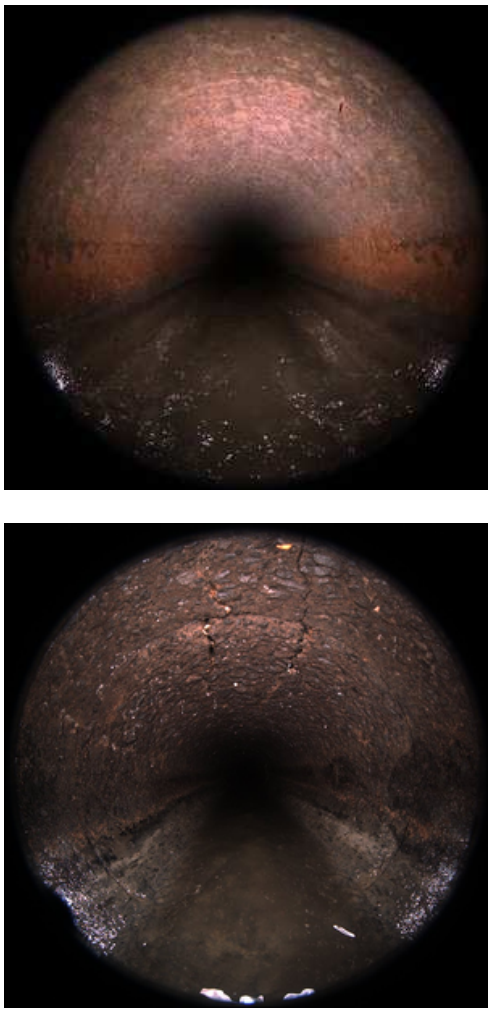


Figure 1: fisheye images captured inside of the pipelines. The pipe on the top is in relatively good condition, whereas the one on the bottom is in poor condition.

little reliable texture. We will therefore make our system capable of handling pipes of this type.

To assess the pipeline condition, images are collected by a mobile camera travelling through the pipelines. To capture more details from the pipe surface, the mobile camera is equipped with a wide view angle fisheye lens rather than an ordinary perspective lens. During photographing the illumination is provided by a light fixed to the camera, which can only illuminate a limited range in the pipe unevenly. Figure 1 shows two example images captured in different pipelines. As we can see, only the peripheral regions of the images contain clear pipe surface. The texture on the pipe surface is fine and weak. In the same pipe the surface appears to be similar everywhere. A sequence of images is captured as the camera moves. Two adjacent images share overlapping regions.

Our reconstruction follows a four-step paradigm.

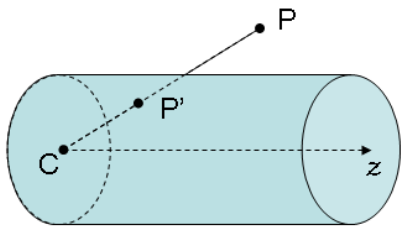


Figure 2: image cylinder: **C** is the camera center; *z* axis is the central axis of this image cylinder; **P** is an object point, with **P'** as its image.

1. Firstly, initial point matching is established between overlapping images;
2. Matched points will then be utilized to estimate the relative pose of calibrated cameras corresponding to different views.
3. With the obtained camera parameters, we implement dense matching between overlapping images while enforcing the epipolar constraints. This step was never included in the previous works [3, 8, 9], in which only those matched points detected in the first step were reconstructed. This step is also arguably the most sophisticated and time consuming step in the whole algorithm. Handling it efficiently is our major contribution.
4. Finally, the 3D location of each point in the image is densely determined through triangulation and a 3D model is built up.

As computer vision algorithms about perspective cameras have already been well studied [4], one might transform each fisheye image to a perspective image to simplify the subsequent process [8, 9]. However, such a transformation either produces an extremely large perspective view which significantly upsampled the peripheral region in the original image, or produces a perspective view of proper size but at the cost of cropping off the peripheral region. In either case, we will destroy the region which really contains the important information in the original image. Therefore, in our work we choose to process the images in their original form, or transform them, when necessary, onto an image cylinder (Figure 2) instead of an image plane.

We define an image cylinder by specifying its central axis. Its radius can be deliberately set to unity without affecting its functionality. The central axis of the image cylinder can be the optical axis of the camera or the baseline between two cameras, depending on the circumstances. The cylindrical image of each point in the 3D world is generated by the intersection of the image cylinder and the ray going from the point to the camera center. Each parallel line on the cylindrical surface

functions like a perspective camera by itself, however, altogether they receive an omnidirectional image more readily than a normal perspective camera does. This image cylinder is particularly useful during point matching and depth estimation, as will become clear soon.

In the remaining part of this paper, we discuss each step mentioned above in detail.

3 INITIAL POINT MATCHING

Due to the development of local invariant features [13, 16], finding corresponding points between overlapping images is much easier now than ever. Comprehensive surveys into the feature detectors and descriptors can be found in [14, 15]. However, point matching on a pipe surface is still difficult due to the faint and similar patterns everywhere. Moreover, whereas all the proposed local invariant features are approximately invariant under affine transformations, the transformation conducted by a fisheye lens is not even perspective, but nonlinear. Thus the corresponding points identified by local invariant features on pipe surface contain many false matches. Our experiments show that the number of false matches can easily exceed the number of true matches by an order of magnitude.

To improve the situation, besides enforcing loose geometry constraints, we transform each fisheye image onto the image cylinder we discussed in Section 2. The image cylinder here takes the optical axis of the camera as its central axis. The consequential advantage of such a transformation is obvious. Since the optical axis of all cameras are roughly parallel to each other as well as to the central axis of the pipe, the images generated on different image cylinders only differ from each other approximately by a simple translation. Comparing to the original fisheye images, we not only remove the scale difference between corresponding regions in different images, but also largely rectify the distortion caused by the nonlinear projection through a fisheye lens. Hence the corresponding points found by local invariant features on the cylindrical images are more reliable. Geometry consistency is also easier to enforce on the transformed images. All line segments connecting corresponding points in two cylindrical images should be roughly parallel and of almost the same length. After detecting corresponding points in the transformed images, we can easily back-project them onto the original fisheye images to facilitate camera pose estimation.

Figure 3 shows the matching results on the original images and the transformed images respectively. Particularly, Hessian-affine detector [16] and SIFT descriptor [13] are used for feature extraction. Matches are identified if two SIFT features share a Euclidean distance under a predefined threshold. Although point matching is between two images, we only present one of them here for clear presentation. We plot the matched points from two images onto one image and connect each pair with a yellow line segment. As we can expect, the lines in the original image should all roughly point to the image center, whereas those in the transformed images should all be roughly horizontal. We only present those matches that can be verified with these loose geometry constraints in Figure 3. On the original image 239 matches passed the verification, whereas on the transformed image 563 matches passed the verification. That justifies our earlier discussion that matching on the transformed images is more reliable.

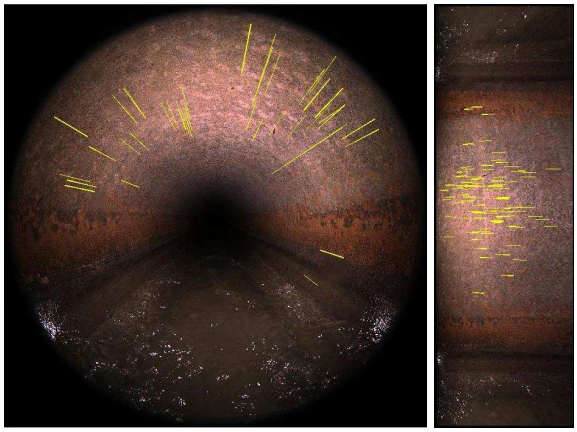


Figure 3: matches found on the image pair of original form and transformed form respectively. Only those matches that pass the loose geometry verification are presented. The rejected false matches are thousands in number. They happen so frequently because the pipe surface is similar everywhere.

Intuitively the matches from both cases are more than sufficient to implement subsequent estimation. One might therefore suspect the necessity of the cylindrical transform. However, as we can see, the lines presented in the image do not seem to match the numbers given above. That is because more than one match can happen intensively on neighbor pixels. Considering matches at the same location does not increase the estimation accuracy, more matches than sufficient is in fact necessary. The number of qualified matches also depends on the texture on the pipe surface. On some smooth surfaces, the number of matches will be much smaller as fewer local features can be detected. That is when the image transformation becomes more important.

Some false matches still remain in Figure 3 as their line segments are not of reasonable length. Again, it is more convenient to enforce this constraint on the transformed images rather than on the original images. On the transformed images, the length of all lines segments should be roughly equal. In the original images, their length should not be equal due to the nonlinearity of the fisheye lens, which is difficult to use as a loose constraint.

4 CAMERA POSE ESTIMATION

A calibration method for fisheye cameras can be found in [10]. Here we assume the camera is calibrated and only aim to estimate the external parameters of the camera. We have briefly discussed the reason why we do not transform the original image into perspective view in Section 2. Particularly on camera pose estimation, the nonlinear transformation between a fisheye view and a perspective view might significantly enlarge the matching error from one pixel to hundreds of pixels in the peripheral region of the image. Hence we need a pose estimation scheme that can be applied directly to fisheye images and is efficient.

We use a modified version of the direct estimation method initially designed for perspective cameras [6]. The main result of the original method is that, given a close enough initialization of the camera parameters as well as the point locations, a structure from motion problem can be solved directly using some iterative optimizing algorithms, e.g. the Levenberg-Marquardt algorithm [12]. The advantage of this method lies in the fact that it is one-stop. It requires no sophisticated operation on any interim variables, e.g. the fundamental matrix required in [5] or the measurement matrix required in [21]. The disadvantage of this algorithm is the requirement of a close initialization, which is usually impossible, especially when the number of unknown parameters is huge.

We discover that the advantage and the disadvantage of the direct estimation method can be magnified and reduced respectively in our problem. In particular, unlike the other algorithms of structure from motion applications, this algorithm bears no assumption on the camera model, neither perspective nor affine. That means it can be adapted to fisheye camera as well, as long as we change the cost function in the Levenberg-Marquardt minimization from the perspective projection to the fisheye projection. Furthermore, as we know the normal condition of the pipelines as well as the approximate location of the camera with respect to the pipe, we can initialize all the parameters accordingly. Obviously, many other inspection purposed applications share the same convenience.

Another important fact about the parameter initialization is that, the parameters are not independent. More precisely, from the parameters of two random cameras, we can accurately determine the 3D locations of all the matched points captured by the two cameras through triangulation. This observation largely reduces the number of variables we need to initialize, i.e. we only initialize the camera parameters, and then derive the location of the points. Besides dependence, obviously, there is also independence between different parameters. Whereas millions of points were captured from thousands of different locations, the camera pose for each image is only related to the dozens of points that have appeared in its image. The location of each point is only affected by the few cameras capturing it. This observation not only leads to the simplification within Levenberg-Marquardt optimization, i.e. the sparse-Levenberg-Marquardt [6], but also to the simplification of our reconstruction. We firstly estimate the camera parameters and points location locally between each pair of adjacent images with the direct estimation method. Although this estimation is local, it has already considered most of the information relevant to the two cameras. Hence the output should still be quite accurate. We then transform all the estimated points and cameras into the same frame of reference. That gives us the initialization of a global direct estimation. Indeed, when the global consistency is not compulsory, we can even terminate without a global estimation. Later we will see, at least for the purpose of pipe condition assessment, local estimation can already detect deformation and cracks on the pipe surface.

The error to be minimized with Levenberg-Marquardt algorithm is given by (1), where $\hat{x}_{ij}$ is the coordinates of point i observed in image j, and x_{ij} is the estimated coordinates of the corresponding point in the corresponding image. When $\hat{x}_{ij}$ is unknown, which really means point i is not observed in image j, we set $\hat{x}_{ij} = x_{ij}$, so that their difference is 0 and the total error will not be affected. During local estimation, as the numbers of points and cameras are limited, the sparse-Levenberg-Marquardt algorithm converges quickly. In our experiment, it takes about 0.5 seconds to estimate the relative pose between each pair of cameras, when 200 point matches are involved. The root mean square error is around one pixel upon converging.

$$e = \sum_j \sum_i ||\hat{x}_{ij} - x_{ij}||^2 \tag{1}$$

Further more, we might add the intrinsic camera parameters into the local estimation. That converts our problem to an uncalibrated reconstruction, requiring inputting three images each time. We do not recommend estimation based on three views. That is because the number of matching points that can survive three views are usually too small to facilitate reliable estimation.

5 INTENSIVE MATCHING

Whereas reconstructing a set of isolated points is sufficient to reveal the pipe deformation on large scale, intensive points reconstruction is required to reveal those cracks which are only several pixels wide on images. To intensively reconstruct the pipe surface, we need intensively match the points on the pipe surface.

Implementing intensive stereo matching between overlapping images is by nature a difficult problem, even though we can narrow the matching range using

the epipolar constraint. A good review of relevant algorithms can be found in [19]. The state of the art of intensive stereo matching lies in the α-expansion method proposed in [22], which approaches the problem by way of optimizing a multi-label Markov Random Field (MRF). However, when the size of the image is huge, optimizing a corresponding MRF requires heavy computation. Another method called FastPD [11] is faster but requires much more memory. More recently, a hierarchical mechanism is incorporated into MRF optimization [23], enabling optimizing large MRFs more efficiently with low memory occupancy.

However, due to the following reasons, our problem cannot be solved by these off-the-shelf methods. Firstly, since the light source is carried by the moving camera, corresponding points in different images are captured under significantly different illuminations, which obviously makes the matching tougher. Secondly, even the hierarchical mechanism [23] largely boosts the speed of solving an individual problem, intensively matching a large number of images is still a huge task. Therefore, we propose two mechanisms to improve the situation.

5.1 Illumination Regularization

Some illumination invariant description and comparison methods have been proposed in the literature, such as the Normalized Cross-correlation (NCC) and the SIFT descriptor [13]. They non-exclusively require more complex computation, which will significantly slow down the system. Here instead of using illumination invariant description, we make the illumination invariant.

Although the light source moves during photographing, its relative position to the camera center is fixed and the location of the camera center within each cross-section of the pipe is in general stable. That suggests, the pipe surface captured by the pixels on the same location within every image is illuminated by approximately the same light. From each pipeline, we have collected thousands of images. The average grey level of a pixel on the same location over thousands of images can be then regarded as the illumination intensity of this pixel or its corresponding points on the pipe surface.

Figure 4 shows the average illumination intensity on images captured in the two pipelines. They are different because the camera travelled at different height in the two pipes and the deterioration degree of the two pipes are different. Based on the illumination intensity images in Figure 4, we can regularize the illumination within each image through (2), where $I(i)$ is the pixel value of pixel i in the original image, $G(i)$ is the grey level of pixel i in the illumination intensity image, a is a positive constant controlling the brightness in the regularized image. Figure 5 compares the image before and after illumination regularization. Especially on the regularized cylindrical images, the obvious illumination variance is removed leaving all pixels under comparable illumination. After illumination regularization, we can easily measure the similarity between pixels by the absolute difference between their regularized pixel values.

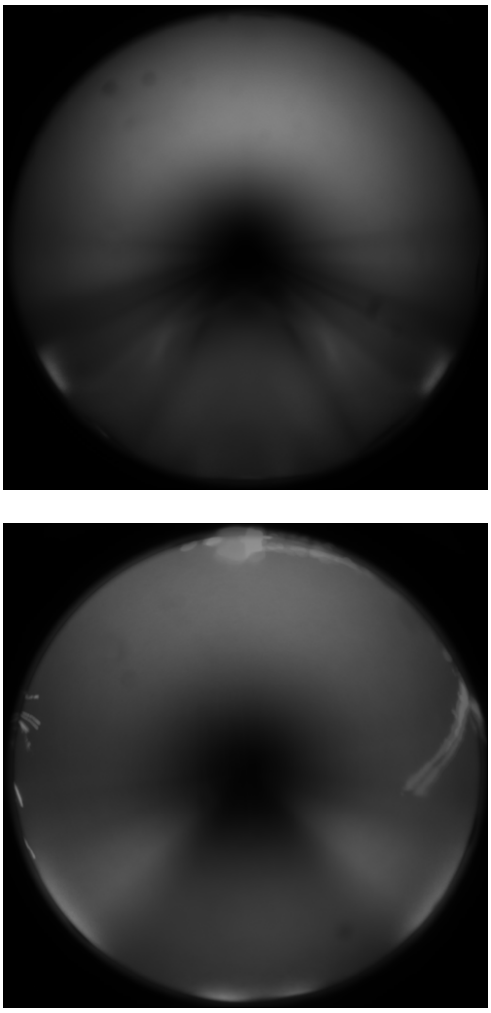


Figure 4: the average illumination intensity obtained from images of two pipelines: some dark blobs can be observed on the top image, which were caused by water drops spread onto the lens; the white threads on the bottom image are caused by some rubbish attached to the lens. However, their affect to the matching process is ignorable.

$$I_r(i) = \frac{aI(i)}{G(i)} \tag{2}$$

5.2 Sequential MRF Optimization

We first explain the design of α-expansion as well as its hierarchical version in our problem and then intro-

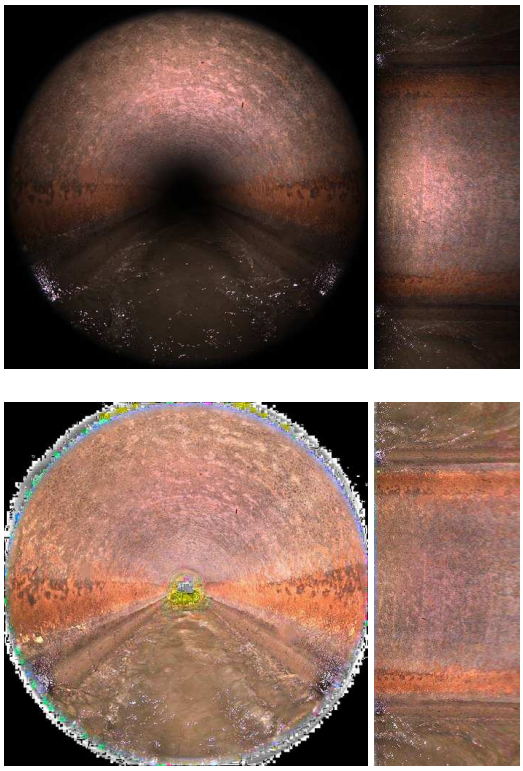


Figure 5: images before and after illumination regularization.

duce our sequential mechanism, which further boosts the speed of our system.

α-expansion After rectification [4], the corresponding points between two overlapping images all lie in corresponding scan lines. One of the two images will later be referred to as the reference. Localizing corresponding points along a scan line, namely estimating the disparity of each object point within the two images can be modelled as estimating the variables in a second order Markov Random Field.

In particular, each variable in the MRF corresponds to a pixel in the reference image. The value of each variable corresponds to the disparity of its corresponding pixel. The probability for each variable to have a particular value, or equivalently for a pixel to have a particular disparity, is subject to two factors. The first one, a function of the color difference between the two pixels related by this particular disparity, is usually referred to as the unary term or the data term. In our work, we use the following unary term:

$$U_i = ||I_1(i) - I_2(i')||_1 \tag{3}$$

where $I_1(i)$ is the color of pixel i in the reference image, $I_2(i')$ is the color of the pixel related to i by its current disparity in the other image, and U_i is computed as a L_1-norm difference between the two. The other one, a function of the disparity difference between the pixel and its neighbor, is usually referred to as the binary term or the smooth term. Each pixel usually has four neighbors, hence there are four binary terms. Binary terms are used to enforce the smooth constraint, i.e. the disparity of points in a scene should be smooth almost everywhere [17]. In our work, we use the following binary term:

$$B_{ij} = |L(i) - L(j)| \tag{4}$$

where $L(i)$ and $L(j)$ is the disparity of two neighbor pixel i and j in the reference image, and B_{ij} is computed as their absolute difference.

The unary term and the binary term really play the role of likelihood and prior in the Bayesian theory. Therefore, through maximizing the probability of a MRF, one really globally maximizes the posterior of each variable and obtains the most probable disparity of each pixel. Due to the Hammersley-Clifford theorem, maximizing the joint probability of variables in the above MRF is equivalent to minimizing the following cost-function:

$$E = \sum_i U_i + \lambda \sum_{ij} B_{ij} \tag{5}$$

where λ is a positive constant balancing the weight between the unary term and the binary term. An effective way of perceiving (5) is through constructing a weighted graph. As shown by Figure 6, each vertex in the graph corresponds to a pixel in the reference image or a disparity value. Edges are created between each disparity vertex and all the pixel vertices. Each edge of this type can be represented by a term U_i in (3). Pixel vertices which are neighbors in the image are connected by edges as well. Each edge of this type corresponds to a B_{ij} in (4). Then, minimizing (5) is equivalent to finding the minimal cut on its graph after which each subgraph contains one and only one disparity vertex.

If the graph contains only 2 disparity vertices, the minimal cut can be found using the max-flow algorithm, regarding the two disparity vertices as the source and the sink respectively. When the number of disparity vertices is larger than two, minimizing (5) is in general NP-hard [2]. α-expansion can provide a high quality suboptimal solution in polynomial time.

Starting from a random initial state, α-expansion sequentially examines the applicability of each disparity, represented by α, to all the pixels. In particular, for each α, a new graph is created. In the new graph, the source node corresponds to the current disparity of each pixel; the sink node corresponds to the α disparity. Those pixels, whose current disparity is α are not included into the new graph. A bi-cut is then implemented using max-flow algorithm to determine whether the pixels currently having other disparities should change their disparities to α. After each

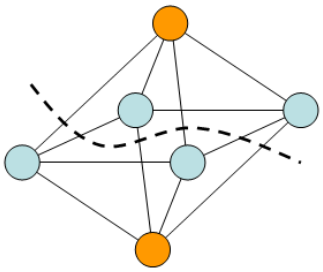


Figure 6: a graphic explanation of minimizing (5): the four blue vertices each correspond to a pixel in the reference image; the two orange vertices correspond to two possible disparities respectively; minimizing (5) is equivalent to a minimal cut to the graph after which each subgraph contains one and only one disparity vertex. The dashed line in the figure shows a possible cut.

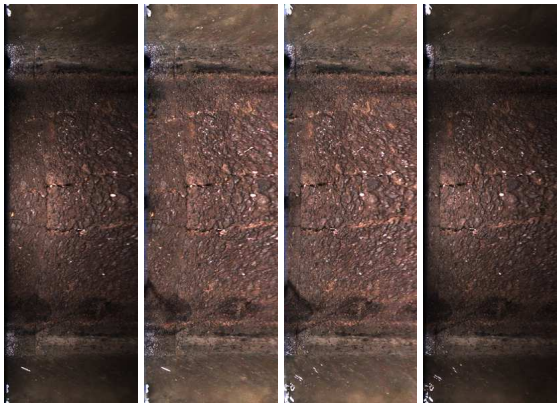


Figure 7: two adjacent images mapped onto the same image cylinder. Images before and after illumination regularization are both provided for comparison.

round of bi-cut, only the subgraph containing α vertex can be increased. That is why the algorithm is named as α-expansion. To compensate the loss in optimality, multiple outer iterations are usually implemented.

Denote the number of outer iterations as m, the number of disparity vertices as n, the processing time of max-flow algorithm as f. The processing time of α-expansion is mnf. When running on images of small size, *e.g.* 300×300, α-expansion can usually terminate quickly in 20 seconds on a normal PC. However, when dealing with a pair of images in large size, whose disparity range is usually large as well, the max-flow algorithm needs to be implemented on a huge graph many times. The processing time of α-expansion expands significantly. Our experiments show that when dealing with a stereo pair in the size of 1000×1000, α-expansion needs more than 30 minutes to converge. That is by definition too slow for practical use. The alternative method, FastPD, cannot be applied either, because a normal PC cannot provide sufficient memory space.

Hierarchical α-expansion The idea of hierarchical α-expansion can be explained as solving the problem with α-expansion under a low resolution first, and then fine tuning the low resolution solution onto higher resolution through optimizing another MRF. More details can be found in [23]. As these two steps can be implemented recursively, the original problem is really solved in a coarse-to-fine manner. Besides, since the MRFs being optimized in the two steps are both much smaller than the original one, the processing speed is largely improved. With the hierarchical α-expansion, processing a stereo pair in the size of 1000×1000 requires only around 10 seconds on a normal PC, and the optimality is comparable to the original α-expansion.

Figure 7 shows two sample images on which we have implemented hierarchical α-expansion. This time, the central axis of the image cylinder is the baseline connecting the two camera centers. As we have already obtained the external parameters of the cameras, we can accurately generate the cylindrical image through back-projection. Although this image cylinder is different from an image plane in shape, it can parallelize the epipolar lines as well. It takes minor effort to snip and unwind that cylindrical image into a planar image. Just make sure to snip the two cylinders along the same epipolar line. So we obtain an image pair in the form people usually deal with during intensive matching, namely corresponding points always lie on the same scan line. The pipe surface presented in these two images contains a vertical connection line and two horizontal narrow cracks, which will test our algorithm's capability in detecting small defects on the pipe surface. We crop off the region submerged by water before implementing graph cuts. That is because we are only interested in the pipe surface, and that dropping the water region can help saving processing time. Figure 8 shows the interim and final results of the hierarchical graph cuts. We can see how the final disparity map is reached through a coarse-to-fine procedure. The disparity value is larger in the center of the image, which corresponds to the top region in the pipe. That suggests that the camera is closer to the top of the pipe compared to the left and right sides of the pipe. The vertical connection line and the horizontal cracks can be clearly observed in the final result as well.

Sequential α-expansion To further boost the processing speed, we propose a sequential mechanism in MRF optimization, the key idea of which lies in better label initializations and smaller label range. The time cost by the max-flow algorithm which is a subroutine in α-expansion depends on the flows needed to be pushed before reaching the optimal state. The number of necessary flows depends on the initial state of the network. That really suggests, if the initial state of the network is

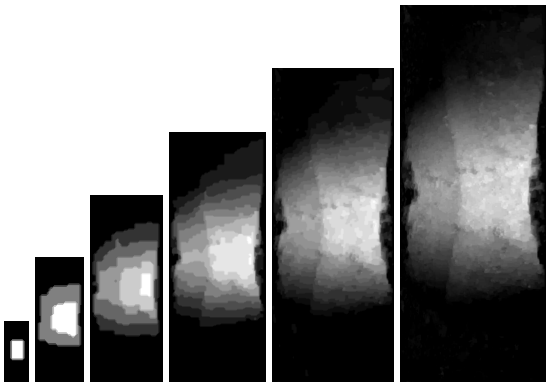


Figure 8: the interim and final results of the hierarchical graph cuts.

more similar to the optimal state, fewer flow, and hence less time, will be needed in optimization. Moreover, starting from an initial state close to the optimal also reduces the number of outer iterations in the α-expansion algorithm. A smaller label range will reduce the number of max-flow implementations in a single iteration.

Whereas for a contextless image pair one can only initialize all labels to be zero or arbitrary values, for sequential pipe images in our case we can largely predict the label configuration. The disparity of each point on the pipe surface is determined by two factors: firstly, its deterioration degree; secondly, and more importantly the location of the camera center. If the camera center travels along the central axis of the pipe, the disparity of different points will only differ slightly due to deterioration. However, if the camera center travels along some line far away from the central axis, the disparity of different points on the pipe will vary significantly. Although the deterioration degree of different regions on the pipe surface is arbitrary, the location of the camera center within each cross section of the pipe is generally stable. Therefore, we only use a large label set during the intensive matching for the first few image pairs. We can then acquire the relative location of the camera within the cross-section of the pipe, or more directly the average disparity along each scan line in the image. On subsequent image pairs, the pixels on each scan line are initialized with the corresponding average disparity. A smaller label set will then be used to estimate their disparities accurately. The smaller label set only needs to cover the variety caused by deterioration, which will be significantly narrower than that caused by the camera location. The MRF optimizing speed is hence boosted.

6 BUILDING A 3D MODEL

Through dense matching on the image cylinder, we have acquired the depth information related to each pixel on the cylindrical image. Together with the camera parameters estimated earlier, we can easily determine the 3D location of each point on the pipe surface through triangulation. The scale ambiguity is removed by setting the length of the baseline between two camera centers as unity. From each pair of adjacent images, we can obtain several millions of isolated 3D points. For better visualization, we might reconstruct a continuous surface with these isolated points using the algorithm proposed in [7]. However, a model containing millions of independent points is too huge for a normal PC to render.

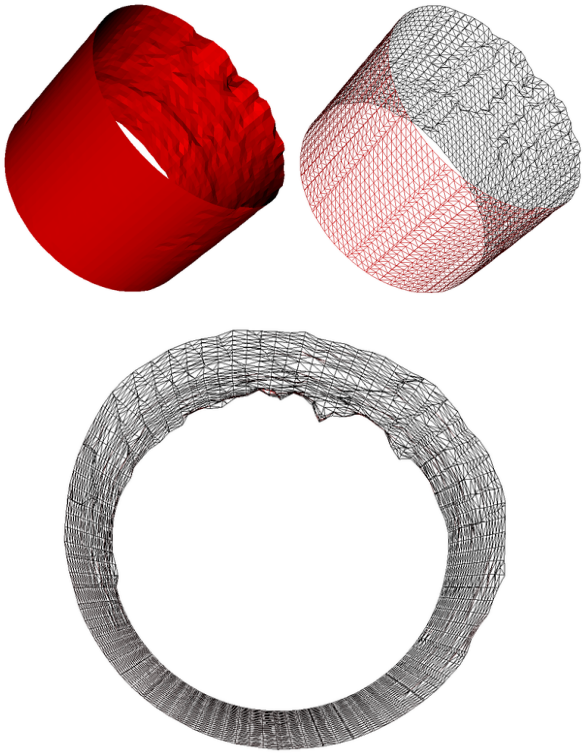


Figure 9: reconstructed pipe surface from the point cloud together with its triangulation state.

Figure 9 only shows the surface reconstructed from one hundredth of all the points. However, even after this significant downsampling, the connection line is still clearly presented, so is the pipe deformation on the large scale. The two cracks are missing because they are both less then ten pixels wide, which can not be preserved during this one hundredth downsampling. However, their existence and state have been represented by the point cloud containing millions of independent points, which will be assessed by civil engineers during force analysis. Note that we can only reconstruct the pipe surface above the water. We observe a complete cylinder here because the missing part has been manually complemented with ideal cylindrical surface.

7 CONCLUSION

We successfully reconstruct the inner surface of buried pipelines from a sequence of fisheye images. The obtained point cloud can be used to generate a virtual surface for visualization, as well as to facilitate other algorithms for pipe condition analysis. We used various efficient and reliable schemes over the four-step reconstruction. We paid particular attention to the process of intensive matching, which is generally slow and memory demanding based on previous algorithms. Our new method overcomes the obstacle of illumination variance and largely boosts the speed. More improvement on 3D model generation is still necessary. One possible development lies in automatically detecting regions of interest and unevenly downsampling the point cloud accordingly. This will be a direction of future work.

ACKNOWLEDGEMENTS

This work was supported by CSIRO, Water for a Healthy Country Flagship.

REFERENCES

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *ICCV*, pages 72 –79, 2009.

[2] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Appl. Math.*, 123:155–225, November 2002.

[3] D. Cooper, T.P. Pridmore, and N. Taylor. Towards the recovery of extrinsic camera parameters from video records of sewer surveys. In *MVA*, pages 53–63, 1998.

[4] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, March 2004.

[5] Richard I. Hartley. Estimation of relative camera positions for uncalibrated cameras. In *ECCV '92: Proceedings of the Second European Conference on Computer Vision*, pages 579–587, London, UK, 1992. Springer-Verlag.

[6] Richard I. Hartley. Euclidean reconstruction from uncalibrated views. In *Applications of Invariance in Computer Vision*, pages 237–256, 1993.

[7] Hugues Hoppe. *PhD Thesis: Surface Reconstruction from Unorgnized Points*. 1994.

[8] J.H. Kannala, S.S. Brandt, and J. Heikkila. Measuring and modelling sewer pipes from video. In *MVA*, volume 19, pages 73–83, March 2008.

[9] Juho Kannala and Sami S. Brandt. Measuring the shape of sewer pipes from video. In *MVA*, pages 237–240, 2005.

[10] Juho Kannala and Sami S. Brandt. A generic camera model and calibration method for conventional, wide-angle, and fisheye lenses. *PAMI*, 28(8):1335–1340, 2006.

[11] N. Komodakis and G. Tziritas. Approximate labeling via graph cuts based on linear programming. *PAMI*, 29(8):1436–1453, August 2007.

[12] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168, 1944.

[13] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[14] K Mikolajczyk and C Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, October 2005.

[15] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65:43–72, November 2005.

[16] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *IJCV*, 60(1):63–86, 2004.

[17] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, September 1985.

[18] Radka Pospíšilová. Occlusion detection and surface completion in 3d reconstruction of man-made environments. In *WSCG*, 2007.

[19] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47:7–42, April 2002.

[20] N. Snavely, S.M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *CVPR*, pages 1 –8, 2008.

[21] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. Comput. Vision*, 9(2):137–154, 1992.

[22] R. Zabih, O. Veksler, and Y.Y. Boykov. Fast approximate energy minimization via graph cuts. In *ICCV*, pages 377–384, 1999.

[23] Yuhang Zhang, Richard Hartley, and Lei Wang. Fast multi-labelling for stereo matching. In *ECCV 2010*, volume 6313, pages 524–537. Springer, 2010.

Deducing Explicit from Implicit Visibility for Global Illumination with Antiradiance

Gregor Mückl
University of Stuttgart
Allmandring 19
70569 Stuttgart, Germany
Gregor.Mueckl@visus.uni-stuttgart.de

Carsten Dachsbacher
Karlsruhe Institute of Technology
Am Fasanengarten 5
76131 Karlsruhe, Germany
dachsbacher@kit.edu

Abstract

The antiradiance method, a variant of radiosity, allows the computation of global illumination solutions without determining visibility between surface patches explicitly, unlike the original radiosity method. However, this method creates excessively many links between patches, since virtually all elements exchange positive and negative energy whose interplay replaces the visibility tests. In this paper we study how and if explicit visibility information can be recovered only by analyzing the link mesh to identify chains of links whose light transport cancels out. We describe heuristics to reduce the number of links by extracting visibility information, still without resorting to explicit visibility tests, e.g. using ray casting, and use that in combination with the remaining implicit visibility information for rendering. Further, to prevent the link mesh from growing excessively in large scenes in the beginning, we also propose a simple means to let graphic artists define blocking planes as a way to support our algorithm with coarse explicit visibility information. Lastly, we propose a simple yet efficient image-space approach for displaying radiosity solutions without any meshing for interpolation.

Keywords: global illumination, radiosity, antiradiance

1 INTRODUCTION

In the 1990s, radiosity methods have been significantly improved, but after a period of large interest research essentially ceased for several years. Mainly three difficulties with radiosity caused the degrading interest: meshing of the input geometry and computing the (hierarchical) link mesh, the necessity of storing all patches and radiosity values in memory for computing a view-independent solution, and above all, the expensive visibility computation that typically consumes most of the computation time [8]. However, some of these problems have recently been successfully tackled: the antiradiance method reformulates the rendering equation such that visibility computation for form factors is no longer necessary, and by this enables a simple and fast GPU implementation of radiosity methods. Dong et al. [4] also demonstrated a GPU-radiosity algorithm by coarsely discretizing visibility that can be computed without ray casting. Motivated by this progress, Meyer et al. [15] introduced a data-parallel method for meshing and hierarchical linking, and demonstrate a CUDA implementation. In combination, these methods allow for interactive radiosity in dynamic scenes. Fig. 1 illustrates the fundamental differences of traditional radiosity, and the two aforementioned improvements. While Dong et al.'s [4] method produces small link meshes – actually smaller than traditional radiosity – it affects the global illumination result negatively due to the coarse discretization. In this paper, we focus our study on the antiradiance method which matches traditional radiosity in terms of quality. However, it replaces costly visibility computation for form factors by creating excessively many links to transport negative energy to compensate for missing occlusion (see Fig. 2).

That is, two solutions at the opposite ends of the spectrum exist for handling visibility in the radiosity method: either fully explicit or fully implicit. This paper bridges the gap in between by presenting a method to deduce explicit visibility information from the link mesh that is generated for the antiradiance (AR) method. By this we can reduce the number of links, however, still without computing form factors with explicit visibility, e.g. using ray casting. Obviously, the visibility information of a scene must be encoded in the AR link mesh: otherwise it would not be possible to compute the correct global illumination result with implicit visibility. Our method starts at exactly this point: we analyze the link mesh to identify chains of links whose light transport cancels out. Once such a chain has been identified, we can remove it without changing the result or reducing the rendering quality noticeably. However, reducing the number of links saves memory and computation time during light prop-

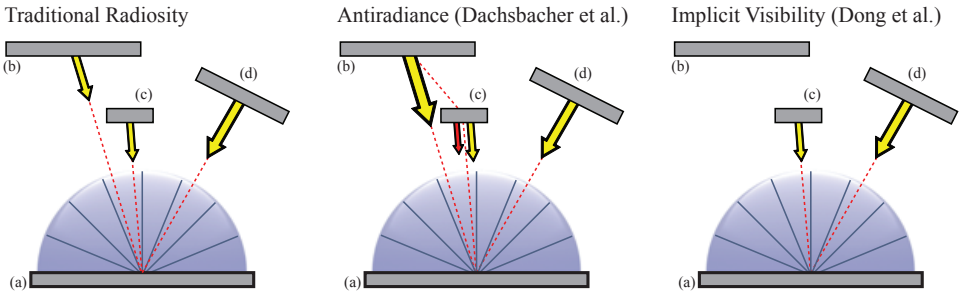


Figure 1: Light transport in different radiosity variants between a patch (a), and three other patches (transport links to (a) shown as dashed lines; sizes of yellow arrows indicate transported intensity): *Left:* radiosity computes visibility for each pair of patches; the transport from (b) is partly blocked. *Center:* the antiradiance method transports energy as if there is no blocking, but compensates for this by propagating negative light that originates from (b) via (c) to (a) (red arrow). *Right:* Dong et al. [4] discretize visibility to one link per direction bin eliminating transport from (b) to (a). In case of full visibility, as for patch (d), all three methods yield the same result.

agation. We consider our contribution being a principal study of deducing explicit from implicit visibility, and report statistical as well as visual results from our prototypal implementation. We also incorporate an effective way to let graphics artists define during the modeling stage where light transport cannot take place. For example, if there is no light exchange between two wings of a building, then we indicate this by simply placing a polygon somewhere in between. Our algorithm will use this coarse explicit visibility information to prevent the AR link mesh from growing large even prior to our reduction. Lastly, we describe a novel way for rendering high-quality antiradiance solutions without meshing or interpolating the final radiance across neighboring elements. At render time, we light the scene using every element as an area light source with an (anti)radiance distribution obtained from the global illumination solution. This allows the rendering of images with high quality interpolation and better contact shadows.

2 RELATED WORK

The importance of global illumination (GI) for computer graphics can be seen from the vast body of research papers in this field. Two main directions have been studied intensively in the last decades: ray tracing and radiosity methods; we refer the reader to excellent text books on these topics, e.g. Dutré et al. [5].

With the increasing computational power of graphics hardware, there have been many attempts to use GPUs to speed up global illumination. In recent years, research in ray tracing made a great leap forwards and there exist algorithms for real-time, parallel kd-tree construction [24], BVH construction [12], and fully interactive GI based on photon mapping [22].

Ray tracing based methods often cache information about the lighting in a scene, e.g. irradiance caching [23], photon mapping [9, 7], or instant radiosity [11]. In particular instant radiosity gained much attraction as represent the lighting of a scene by a set of virtual point lights, and thus easily maps to GPUs [18]. The light cuts method [21] clusters point lights into a hierarchy to speed up rendering. Final gathering is an essential step for computing high-quality images and a parallel algorithm therefor has recently be demonstrated [17].

In the following, we discuss work which is more closely related to our approach. There have been several attempts to compute radiosity solutions on the GPU. The main cost factor is the evaluation of the mutual visibilities between surfaces patches. Either rasterization together with the hemicube method [2, 1], or ray tracing on the GPU [19] have been used to compute form factors. The antiradiance reformulation [3] of the rendering equation [10] replaces explicit visibility by a recursive computation with negative light ("antiradiance"). Dong et al. [4] use a directional discretization and store only one link to the respective closest patch per direction. Thus the visibility is constructed implicitly with the link hierarchy. Although both methods are fundamentally different, both rely on a hierarchical link structure which initially had to be generated sequentially on the CPU. Meyer et al. [15] present a

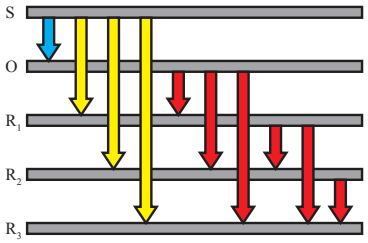


Figure 2: Radiance and antiradiance links for a 1D example. The normal of patch S is pointing downwards, the other surface normals are pointing towards S. There is only one unoccluded radiance link $S \xrightarrow{(+)} O$ (blue). The other radiance links $S \xrightarrow{(+)} R_n$ (yellow) and antiradiance links $R_n \xrightarrow{(-)} R_m$ (red) only exist for implicit occlusion handling.

data-parallel algorithm for link and patch creation, implemented in CUDA, which allows interactive radiosity methods with fully dynamic scenes.

Typically, the per-patch radiosity values that represent the GI solution are interpolated across adjacent patches. This can be done by generating an accordingly tessellated triangle mesh together with Gouraud shading [5] and improved using discontinuity meshing [14]. Dachsbacher et al. [3] used an image-space splatting approach that does not require a special mesh. Lehtinen et al. [13] directly compute global illumination using a meshless hierarchical representation and display the solution similarly.

3 ANTIRADIANCE

In this section we briefly review the antiradiance method [3], also following the notation of this work. The rendering equation describes the equilibrium of light transport in a scene and the radiance at a surface point x in direction ω_o is:

$$\begin{aligned} L(x,\omega_o) =& E(x,\omega_o) + \int_{\Omega^+} f(x,\omega_o,\omega_i) L_{in}(x,\omega_i) \cos\theta \, d\omega_i \\ =& E(x,\omega_o) + (\mathbf{K}L_{in})(x,\omega_o) \; . \end{aligned}$$

The incoming radiance at position x from direction ω_i originates from the closest surface in that direction. It is determined using the ray casting operator $ray(x,\omega_i)$ and part of the transport operator $\mathbf{G}$:

$$L_{in}(x,\omega_i) = L(ray(x,\omega_i),\omega_i) = (\mathbf{G}L)(x,\omega_i) \; . \quad (1)$$

As the computation of $\mathbf{G}$ is a very costly, the AR method strives to replace $\mathbf{G}$ by another transport operator that is cheaper to compute. Instead of resolving visibility explicitly by finding the nearest surface, the radiance is gathered from all surfaces along a ray yielding the transport operator $\mathbf{U}$. Extraneous light is then propagated and must be compensated. A pass-through operator $\mathbf{J}$ is defined that lets incoming radiance at a patch pass through without changing its magnitude or direction. The operators are related as follows:

$$\mathbf{G}L = \mathbf{U}L - \mathbf{U}\mathbf{J}\mathbf{G}L = \mathbf{U}(L-A) \quad (2)$$

with $A = \mathbf{J}\mathbf{G}L$ being the *antiradiance*. With this reformulation of the standard transport operator, the antiradiance rendering equation is obtained as:

$$L = E + \mathbf{K}\mathbf{U}(L-A) \quad (3)$$

$$A = \mathbf{J}\mathbf{U}(L-A) \; . \quad (4)$$

When L, A and E are projected into a suitable Hilbert base over the scene surface with finite dimensionality, these functions can be expressed as vectors of their components in that Hilbert space. Likewise, the operators $\mathbf{U}$, $\mathbf{K}$ and $\mathbf{J}$ become matrices:

$$\begin{pmatrix} L \\ A \end{pmatrix} = \begin{pmatrix} E \\ 0 \end{pmatrix} + \begin{pmatrix} \mathbf{K}\mathbf{U} & 0 \\ 0 & \mathbf{J}\mathbf{U} \end{pmatrix} \begin{pmatrix} L-A \\ L-A \end{pmatrix} \; . \quad (5)$$

We can then separate K out of this matrix and get:

$$\begin{pmatrix} L \\ A \end{pmatrix} = \begin{pmatrix} E \\ 0 \end{pmatrix} + \begin{pmatrix} \mathbf{K} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{U}^L & 0 \\ 0 & \mathbf{J}\mathbf{U}^A \end{pmatrix} \begin{pmatrix} L-A \\ L-A \end{pmatrix} \; . \quad (6)$$

The thus remaining matrix describes the occluded light transport by means of the transport operator $\mathbf{U}$. Replacing it with $\left(\begin{smallmatrix} \mathbf{G} & 0 \\ 0 & 0 \end{smallmatrix}\right)$ yields the discretized equation for occluded light transport (as in standard radiosity) again. This comparison shows that the upper left part $\mathbf{U}^L$ of the matrix describes radiance transport while the lower right part $\mathbf{U}^A$ describes antiradiance transport.

4 REMOVING OCCLUDED LINKS

Eq. 6 shows that once we separated out the transport operator matrix we can interpret antiradiance and traditional radiosity as two extremes of how light is propagated between elements in the scene. However, the transport matrix does not have to take the form of either the fully occluded or the fully unoccluded transport. Within limits that we will discuss in the following, it is possible to create intermediate matrices $\mathbf{U}'^L$ and $\mathbf{U}'^A$ that contain transport with and without explicit visibility, i.e. essentially a mixture of entries from $\mathbf{G}$ and $\mathbf{U}$.

Let us first assume the case where we replace one unoccluded transport by a transport with explicit visibility. For this we define the matrix $\mathbf{U}'^L$ which contains one entry of $\mathbf{G}$, i.e. $\mathbf{U}'^L_{kl} = \mathbf{G}'^A_{kl}$, and all other entries are equal to $\mathbf{U}^L$. If the resulting light transport is correct, then the solutions L_{ij} and L'_{ij} for both matrices are equal. The equation for the k-th patch, L_k, becomes:

$$L_k = \sum_{i \neq l} \mathbf{K}_{ki}(\mathbf{U}^L_{ki}L_i - \mathbf{U}'^A_{ki}A_i) + \mathbf{K}_{kl}(\mathbf{G}_{kl}L_{kl} - \mathbf{U}'^A_{kl}A_{kl}) \; . \quad (7)$$

An entry $\mathbf{G}_{ij}$ is always less or equal to the respective entry $\mathbf{U}_{ij}$, and thus the sum over all $\mathbf{U}'^A_{ki}A_i$ in this equation must either be equal or less than the sum over all $\mathbf{U}^A_{ki}A_i$. This means, that *at least* one of the entries in this particular row of $\mathbf{U}'^A$ must be decreased in value. In other words, if the radiance transport between two patches is performed with proper occlusion, the receiving patch must no longer receive the same amount of antiradiance that was previously transported to it (to account for the occlusion along this transport path that we now consider explicitly). Note that although this shows that unoccluded and occluded light transport can be performed at the same time, no rules for the adjustments to $\mathbf{U}'^A$ can be derived from these equations alone. If we assume for now that we can replace values of $\mathbf{U}^L_{ij}$ one after another, then we can repeat this until $\mathbf{U}'^L$ equals $\mathbf{G}$, and in this case $\mathbf{U}'^A$ vanishes.

4.1 Link Removal in 1D

We have shown that mixing occluded and unoccluded light transport operations is possible under the restriction that antiradiance must only be transported to

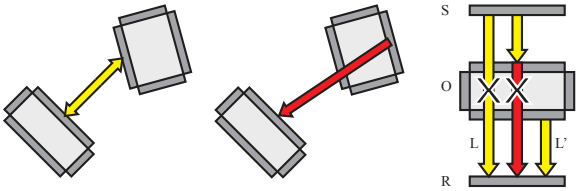


Figure 3: Convention for creating links: radiance links are created between patches facing each other (left); antiradiance links are in the negative hemisphere of a patch (center). Right: For every incoming link (here: L) we search if there are shorter incoming links (L'). Then we can remove L. Note that the antiradiance link shown in red is removed because of the same fact and thus the result is again correct at last.

patches that are the target of unoccluded transport operations. To introduce our algorithm we start with the instructional example of patches along one line. We will discuss rules for removing transport links for the case where a target patch is fully occluded from the source. Note that this information is solely based on the patch positions and extents, and the link mesh. Detecting partial occlusion, i.e. not only removing links for fully occluded patches, requires visibility computation, e.g. using ray casting that we still want to avoid. Also, we only consider opaque surfaces as potential occluders.

The motivation for the link removal is that the implicit handling of visibility in antiradiance generates a number of light transporting links that rises disproportionally with the depth complexity of the scene.

A simple set of surfaces in a 1D example as seen in Fig. 2 motivates the removal of unnecessary links. With unoccluded light transport, the topmost surface S illuminates all other surfaces. Therefore, it is linked to all of these surfaces, although the nearest surface O already fully occludes the light, and the light transported across the links to the surfaces further away needs to be compensated for. This is achieved by the antiradiance links from O to all other patches R_n below, and the pattern repeats analogously for every surface. When explicit visibility is taken into account, only the link $S \xrightarrow{(+)} O$ is required to illuminate this example scene correctly. With the implicit handling of occlusion as described above, $n+n!-1$ excess links are generated for n patches.

The first observation here is that all occluded radiance links in this example intersect the sole unoccluded patch O. Removing them along with all antiradiance links that originate from O itself effectively results in the occluded transport again. However, the remaining antiradiance links do not transport energy anymore since the patches where they start from do not receive any. To optimally reduce the link mesh, we also want to remove those from the transport matrix.

It is now possible to formulate two rules to find

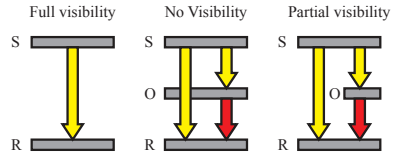


Figure 4: Classification of visibility in 3D used for our heuristics. A sender patch S emits radiance towards a receiving patch R, which may be blocked by a potential occluding patch O. Three situations need to be distinguished: full (left), none/fully occluded (middle) and none (right).

groups of links that can be removed from the link mesh without changing the result:

Rule 1 Search for a pattern of three links: from the sender S to the occluding receiver O, from S to the occluded receiver R, and from O to R. When such a pattern is found, both links to the occluded receiver $S \xrightarrow{(+)} R$ and $O \xrightarrow{(-)} R$ may be removed. We distinguish radiance and antiradiance links by the sign over the arrow.

Rule 2 Check for every *incoming link* of a patch L if there is another, shorter incoming link L'. If such a link is found, we can remove L as it is occluded (Fig. 3). Note that this rule is only valid if the objects in the scenes are manifolds with closed surfaces as also assumed in [3].

4.2 Heuristics for Link Removal in 3D

Obviously, the aforementioned removal in 1D retains validity in two or three dimensions only if it is applied to infinitely small patches along a single ray through the scene. For patches with finite size and only a finite number of discrete directions (*directional bins*), as used in the antiradiance method [3], we can derive heuristics from these rules that still work well when the scene discretization is reasonably fine. In Section 6 we evaluate the validity of both heuristics described in this section.

The modification compared to the 1D case is necessary due to the fact that the visibility function V between two patches is no longer either 0 or 1, but can take any value in-between (see Fig. 4). Furthermore, a special treatment is required to respect peculiarities in a hierarchical link mesh: to determine if the light transport between two patches is blocked, we might have to search across different levels of the hierarchy.

Heuristic 1 The first rule from Section 4.1 transforms into Algorithm 1. Again we find a combination of sender S, occluder O, and receiver R. However, we only remove the links, if O or one of its parents (in the patch hierarchy) subtends a large enough solid angle such that the light transport from a sender to another surface is completely blocked (see Fig. 5). In addition, the radiance link $S \xrightarrow{(+)} R$ must have the same (discrete) direction as the antiradiance link $O \xrightarrow{(-)} R$. To test this,

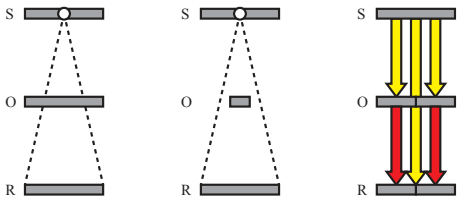


Figure 5: Left: the occluder O, as seen from S (note that in the antiradiance method links are established between patch centers), subtends a larger solid angle than the receiver patch R. Center: the subtended solid angle of O is too small and the link from S to R cannot be removed. Right: the radiance link $S \xrightarrow{(+)} R$ is on a different level of the hierarchical link mesh than the antiradiance links $O \xrightarrow{(-)} R$. Thus we also consider parents in the patch hierarchy when searching for blockers.

we construct two infinite circular cones with their tips in S: the first one, C_{Link} connects the centers of S and O with its axis and has an opening angle so that the solid angle subtended by this cone equals that of O as seen from S. The second cone, C_{bin} has the discrete direction of the link's source bin as its axis and its opening angle is chosen so that C_{bin} covers a solid angle of ω_{bin}. Then, C_{Link} must enclose C_{bin} (cf. Alg. 1 lines 6-10). This ensures that all patches lie on one line, analogous to the 1D example before. When using hierarchical link meshes, an occluder can be linked to the receiver on a finer level than the sender-receiver link, in which case the patch at the finer level takes on the role of R (Fig. 5, right, Alg. 1 line 5). Also, the full extent of the occluding geometry can be better estimated by checking the solid angles subtended by any parents of the suspected occluding patch.

Heuristic 2 The second removal rule transfers to the 3D case analogous to the previous one (see also Algorithm 2). The aforementioned restriction to scenes consisting entirely of closed manifolds stays valid. When

Algorithm 1 Find and disable all links between senders, occluders and receivers.

```
1  for each radiance link L
2    required[L] = true
3
4  for each radiance link L
5    for all target patch P_j of link L and its parents
6      C_Link ← cone(center(P_i), actualdir(L), ω_{i←j})
7      // sourcedir(L) is discreet source direction of Link L
8      C_bin ← cone(center(P_i), sourcedir(L), ω_bin)
9      if ω_{i←j} > ω_bin and C_Link encloses C_bin
10       binOccluded ← true
11
12   if binOccluded
13     for all antriadiance links L' starting from P_j
14       P_k ← target_patch(L)
15       if link P_i → P_k exists
16         if sourcedir(L)=sourcedir(L')
17           if targetdir(L)=targetdir(L')
18             required[L] ← false
19             required[L'] ← false
20
21 remove all links where required[L] is false
```

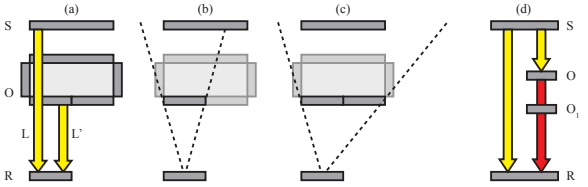


Figure 6: (a) Two incoming links L and L' at R have the same direction. To determine if we can remove L, we need to test if the transport from S to R is blocked. Testing this blocking with the patch O where L' emanates from does not reveal this information. Ascending the hierarchy and testing with the parent of O determines full blocking (c) and L can be faithfully removed. (d) A situation where a fully occluded antiradiance link must not be removed: The link $S \xrightarrow{(+)} R$ is partially occluded by O and O_1 and the link $O \xrightarrow{(-)} R$ is fully occluded by O_1, but the antiradiance produced at O must arrive at R to correctly reproduce the partial occlusion. Therefore, the link $O \xrightarrow{(-)} R$ must not be removed.

we test if a link $S \xrightarrow{(+)} R$ can be removed, we need to find a shorter link from a potential occluder O to R. However, these two links must have the same (discretized) direction (Alg. 2 line 4). If we detect such a situation, then we need to determine if O, or the surface to which is belongs, is large enough to block the transport from S to R by ascending the hierarchy from O to its largest parent. Next, we project S onto the plane of this parent patch and only if the projection is fully on that surface the link can be removed (Fig. 6, Alg. 2 lines 9-15). Note that patches that share a common parent with S will never be considered as occluders (Alg. 2 line 8).

However, not all links that are detected as fully occluded may actually be removed. As illustrated in Fig. 6 antiradiance links might be necessary to capture partial occlusion although the above heuristic marks them as occluded. Algorithm 3 detects these links and prevents them from being removed. For each partially visible link $S \rightarrow R$ it searches all other antiradiance links

Algorithm 2 Classify Link visibility as seen from target patch.

```
1  for each radiance link L
2    visibility[L] = full
3    for each link L' ending at target patch of L
4      if target_bin(L)=target_bin(L')
5        R ← target_patch(L)
6        S ← source_patch(L)
7        O ← source_patch(L')
8        if S and O do not have same parent patch
9          // If S or O are clusters, the following is
10         // checked on every pair of patches in these clusters
11         O' ← biggest parent of O
12         S' ← perspective projection of S onto O'
13         if S on far side of O' and S' fully inside O'
14           visibility[L] ← none
15           continue
16         else
17           visibility[L] ← partial
```

Algorithm 3 Finding required occluded links.

```
1  for each link L
2    if visibility[L] = none
3      required[L] = false
4    else
5      required[L] = true
6
7  for each link L with visibility[L]=partial
8    S ← source_patch(L)
9    R ← target_patch(L)
10   for R and each child patch of R
11     for each antiradiance link L'
12       S' ← source_patch(L')
13       R' ← target_patch(L')
14       if R=R'
15         for each link L'' connecting S to S'
16           if visibility[L''] != none and L'' is radiance link
17             required[L'] ← true
```

$S' \xrightarrow{(-)} R$ to the same target patch and checks if a non-occluded link $S \rightarrow S'$ exists. If so, $S' \xrightarrow{(-)} R$ is required to keep the partial occlusion of S' from R by S intact and is marked as required. Note that this heuristic cannot be used after heuristic 1 because links $S' \xrightarrow{(-)} R$ may already have been removed, resulting in required links getting missed and removed.

Patches without incoming links In case of static direct lighting in the scene we can also remove all links emanating from patches that are neither light sources nor have any incoming links, as these links will never transport energy. However, similar to the original antiradiance implementation [3] we typically use shadow maps for computing direct illumination and thus potentially every patch can receive energy that has to be propagated further.

4.3 User-Defined Link Removal

In addition to the link removal heuristics described above, we can optionally perform link removal based on user-defined (invisible) blocking geometry which strictly cuts all links that intersect it. This additional geometry can be a simple polygon generated by the user along with the scene to separate parts of the scene that obviously do not directly exchange light with each other, e.g. two rooms separated by solid walls (see Fig. 8). Since this geometry consists of few polygons only, we test for every link if it intersects the blocking geometry without generating high cost, and remove the link if this is the case. Note that this is similar to Fradin et. al. [6] who used manually placed portals to section large scenes.

5 FINAL SHOOTING

Dachsbacher et al. [3] used a splatting approach, similar to point-based rendering, to render an image with interpolated patch colors. Instead we propose to use a "final shooting" approach: we treat each patch in the scene as a *patch light source* (PLS) with a directional intensity distribution according to the total exitant intensity determined by the antiradiance solver. For rendering the final solution, we simply light the scene only using the PLSs. This approach can be seen as a variant of instant radiosity [11], where light sources emit radiance and antiradiance (computed using the antiradiance method) and thus account for shadowing implicitly. Note that the resulting number of virtual light sources in our approach is typically orders of magnitudes higher. Furthermore, every PLS is an area light source from which lighting computation is more intricate than from point lights. To this end, when computing the lighting of a fragment due to a PLS, we replace the PLS by 8 point light sources randomly placed on the PLS. This can be seen as a Monte-Carlo sampling of the area light sources; no noise is visible in the images, as the number of PLS is very high (it equals the number of patches in the scene). Note that by lighting the scene with all PLSs we obtain not only a smooth interpolation, but also one (additional) indirect bounce at the same time with no additional cost.

When using a full link mesh, we efficiently accumulate the contributions of all PLSs using deferred shading and interleaved sampling [20]. However, care has to be taken when using final shooting together with the link removal heuristics: in this case, only those patches are to be lit by a PLS that are still linked to it after reducing the link mesh. To account for this, instead of using deferred shading, we have to render every receiver patch for every PLS, compute per-pixel lighting, and accumulate the contributions. Note that this process benefits from the GPU's early-z culling automatically omitting occluded receivers. For lighting computing from a PLS, we look up the interpolated intensity towards a fragment using precomputed interpolation weights in a cube map (6×512^2 resolution in our examples).

6 RESULTS AND DISCUSSION

In this section we compare the heuristics and the user-defined link removal. To determine the impact of the heuristics' approximation regarding the link removal, we modified both to perform explicit visibility checks using ray casting and Monte Carlo sampling instead of the link mesh based checks. Note that no heuristic *should* remove more links than those removed with explicit visibility testing. We have implemented an Antiradiance solver similar to the one described in [3], using the same algorithm for building the hierarchy. The preprocessing, including the previously discussed link removal heuristics, is initially implemented and executed on the CPU (running on multiple cores), while OpenGL is used for the simulation of the light transport and for displaying the result. While our current implementation can be seen as an experimental prototype, we plan to integrate our heuristics into the data-parallel link generation method by Meyer et al. [15] in the future.

			heuristic 1				heuristic 2			
scene	patches	links	explicit test	heuristic removal	incorrect	not removed	explicit test	heuristic removal	incorrect	not removed
Japan	12745	629665	157449	129313	54872 (42%)	83058 (52%)	71007	34408	11732 (34%)	48331 (68%)
Office	14246	1470440	581768	395592	124232 (31%)	310408 (53%)	260475	85075	28739 (34%)	204139 (78%)
Desks	14396	1465632	759669	280705	7752 (10%)	486716 (40%)	690145	300316	11577 (4%)	401406 (58%)
Soda Hall	25023	2774452	2076609	1621749	73998 (5%)	528858 (25%)	1888014	1016923	21026 (2%)	892117 (47%)

Table 1: Results of applying our heuristics to the test scenes with: the total number of radiance and antiradiance links, the number of links removed by the explicit test, the links removed by the heuristic, the number of incorrectly removed links compared to the explicit test, and the number of links that have not been removed by the heuristic but by the explicit test.

scene	patches	links	heuristic 1	heuristic 2
Japan	12745	629665	22.1s	26.0s
Office	14246	1470440	248.2s	224.9s
Desks	14396	1465632	80.5s	127.0s
Soda Hall	25023	2774452	87.9s	280.0s

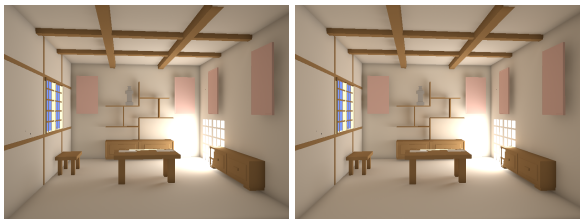
Table 2: Measured run time for our multithreaded CPU implementation of the heuristics running 8 concurrent threads (averaged over 10 runs)

6.1 Link Removal Heuristics

We tested our link removal heuristics against the explicit visibility oracle on various scenes: the Japanese room and office from [3], the "desks" shown in Fig. 8 and the model of the fifth floor of the Soda Hall (see Fig. 12). The results are summarized in Table 1 and the run times are given in Table 2. All measurements were taken on an Intel Core i7 CPU with 2.66MHz, 6GB RAM, and an NVIDIA GeForce GTX 465 with 1024MB RAM, running Linux.

Our results show that the heuristics also remove links that are not totally occluded. In the Japanese room and office scene, which both consist of a single room with many objects inside, the error rate is particularly high while it is low for the "desks" scene and the Soda Hall with their walls as main occluders. This shows that while both heuristics perform well with big solid occluders, they are prone to inaccuracy with correctly estimating visiblity along silhouettes of objects. The first heuristic relies on the subtended solid angle for testing blocking. However, this gives no indication about the actual patch shapes: for instance, elongated patches can be visible although nearly quadratic patches with larger solid angles are in front of them. The second heuristic erroneously removes links for which unblocked paths between the patches may still exist although the projection test succeeds. The runtime for heuristic 1 depends on the number of occluders in the scene: the algorithm only needs one occluder per link and thus has to search less links and terminates quicker when many large occluders are present. Heuristic 2 spends most time in algorithm 3, whose complexity is dominated by the number of links that have to be searched.

Although the resulting error due to the link removal heuristics seems significant for the affected scenes, the impact on the rendered images is hardly perceivable (see Fig. 7 and Fig. 10a). Obviously, errors are only introduced for links whose contribution (either radiance or antiradiance) is negligible. The first heuristic creates generally brighter shadow regions. It discards $O \xrightarrow{(-)} R$



(a) heuristic 1 (b) heuristic 2

(c) 4× difference heuristic 1 (d) 4× difference heuristic 2

Figure 7: The Japan scene rendered with reduced link meshes using final shooting (top), and then luminance differences to an image computing with the full link mesh (red is less, green is greater than original). The difference images have been scaled by a factor of 4. Compare Fig. 9, 10a for references.

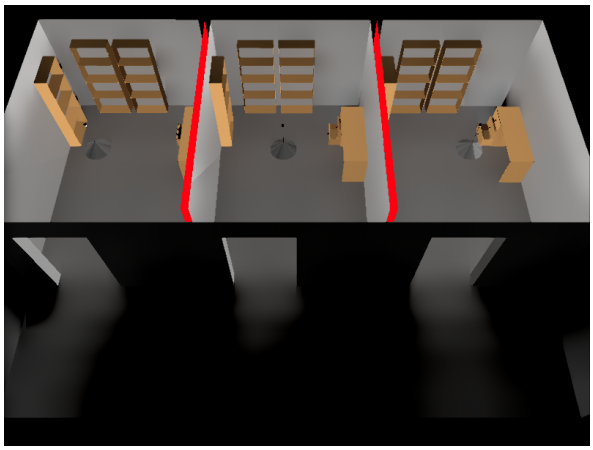


Figure 8: Desk scene with blocking geometry (red) after 6 antiradiance iterations, rendered with splatting.

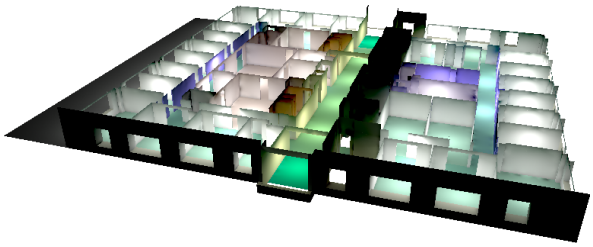


Figure 12: Soda Hall level 5 lit by diffuse emitting area light sources in the rooms and on the hallways. It was rendered using final shooting and the reduced link mesh produced using heursitic 1 (see Table 1).

links even if there exists another link $S' \xrightarrow{(+)} R$ that is only partially occluded by O and requires the antiradiance generated at O to correctly light R. The second algorithm of heuristic 2, which is run to preserve partial occlusion, results in a stronger tendency to keep antiradiance links. Thus, the corresponding difference image shows lesser brightening of shadow regions. Applying heuristic 2 followed by heuristic 1 on the Japanese room and Soda Hall scenes results in 144983 removed links (45.5% error) and 1628987 removed links (4.7% error), respectively, and visual quality comparable to Fig. 7.

We tested the blocking planes by adding two such polygons between the rooms in the "desks" scene (see Fig. 8). These two quads alone caused a removal of almost 76% of all occluded links in the scene, demonstrating that this mechanism is not only cheap to compute, but also highly efficient.

6.2 Rendering Quality

Fig. 9 shows resulting images for the Japanese room without interpolation, with splatting, and final shooting. The global illumination solution has been computed with 4 antiradiance iterations and 128 direction bins for all of our test scenes.

At a resolution of 800×600 pixels the rendering speed was 8.5 frames per second with no interpolation, 7.7 fps with splatting, and 0.04 fps with shooting. Although final shooting has a considerable impact on performance due to the high number PLSs, the resulting images capture finer details, e.g. contact shadows, due to patches emitting antiradiance. Interleaved sampling for final shooting with 4×4 interleaved sampling runs at 0.44 fps, and the performance increases further when using larger interleaving patterns yielding 1.22 fps at 8×8, and 2.88 fps at 16×16. However, at some point the wider Gaussian blur filter again removes details (see Fig. 10). Nichols et al. [16] report tremendous speedups with multi-resolution splatting compared to interleaved sampling, however, there is another way to speed up shooting. So far, we used the leaf nodes in the hierarchy as PLSs, but we also can use interior nodes therefor. Using the leaves' parents yields about 75% less PLSs, but only slightly reduces the amount of detail in the lighting as we treat them as area lights (see Fig. 11), but already yields a significant speedup: shooting at full resolution runs at 0.14 fps, at 1.29 fps with 4×4, at 2.99 fps with 8×8, and at 5.09 fps with 16×16 interleaving, respectively. A further optimization, and direction for future work, is to exploit the patch hierarchy also for selecting the PLSs, in spirit of [21].

7 CONCLUSIONS

In this paper we studied heuristics operating directly on the hierarchical link mesh of the antiradiance method to deduce explicit visibility information and thus to reduce the number of links. The energy propagation is the most time consuming step in antiradiance methods and greatly benefits from link removal as its cost is proportional to the number of links. Our heuristics remove links more faithfully than Dong et al.'s method [4] yielding results of similar quality as the antiradiance method. The user-defined blocker geometry can further be used to remove links with little computation. The final shooting simplifies the rendering of a high-quality final solution and yields better results and requires less tweaking than Dachsbacher et al.'s splatting approach.

Obviously, our heuristics will have to be incorporated into a data-parallel link generation method, such as Meyer et al.'s work [15] to actually speed up antiradiance in fully dynamic scenes. This data-parallel algorithm is about two orders of magnitude faster than our (and Dachsbacher et al.'s) CPU-based link generation algorithm (45 million links/s versus 140 thousand links/s), and we would expect a similar speedup for the link removal. Another challenge is to create blocking geometry automatically by analyzing the scene geometry. We believe that these two steps will enable our link removal in fully dynamic scenes at interactive speed.

ACKNOWLEDGEMENTS

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

REFERENCES

[1] Attila Barsi, László Szirmay-Kalos, and Gábor Szijártó. Stochastic glossy global illumination on the gpu. In *Spring Conference on Computer Graphics 2005*, pages 187–193, 2005.

[2] Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on graphics hardware. In *Graphics Interface 2004*, pages 161–168, 2004.

[3] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 26(3):61, 2007.

Figure 9: The Japanese room with different rendering methods. The room is lit from outside the window on the left. The images were rendered with 4 iterations at a resolution of 800×600 pixels with a full link mesh.

(a) PBRT reference (b) 4×4 interleaved sampling (c) 16×16 interleaved sampling

Figure 10: Results with different interleaved sampling patterns: larger patterns result in stronger blurring and loss of small scale features.

(a) final shooting (b) 4×4 interleaved sampling (c) 16×16 interleaved sampling

Figure 11: Final shooting using not the leafs of the patch hierarchy, but its parents as PLSs. This results in significantly faster rendering speed, sacrificing only little details in the lighting (e.g. visible in shadowed areas around the bookshelf).

[4] Zhao Dong, Jan Kautz, Christian Theobalt, and Hans-Peter Seidel. Interactive global illumination using implicit visibility. In *Proc. of Pacific Graphics*, pages 77–86, 2007.

[5] P. Dutré, K. Bala, and P. Bekaert. *Advanced Global Illumination*. AK Peters, 2006.

[6] D. Fradin, D. Meneveaux, and S. Horna. Out-of-core photon-mapping for large buldings. In *Proceedings of Eurographics symposium on Rendering*, June 2005.

[7] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8, 2008.

[8] Nicolas Holzschuch, François X. Sillion, and George Drettakis. An efficient progressive refinement strategy for hierarchical radiosity. In *Photorealistic Rendering Techniques (Eurographics Workshop on Rendering)*, pages 357–372, 1994.

[9] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2001.

[10] James T. Kajiya. The rendering equation. *Computer Graphics (Proc. of SIGGRAPH '86)*, 20(4):143–150, 1986.

[11] Alexander Keller. Instant radiosity. In *SIGGRAPH '97*, pages 49–56, 1997.

[12] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, 28(2), 2009.

[13] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. A meshless hierarchical representation for light transport. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 27(3):1–9, 2008.

[14] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, 1992.

[15] Quirin Meyer, Christian Eisenacher, Marc Stamminger, and Carsten Dachsbacher. Data-Parallel, Hierarchical Link Creation. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, 2009.

[16] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. In *Computer Graphics Forum 28(4)*, pages 1141–1149, 2009.

[17] Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*, 28(5), 2009.

[18] Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*, 27(5), 2008.

[19] Arne Schmitz, Markus Tavenrath, and Leif Kobbelt. Interactive global illumination for deformable geometry in cuda. *Computer Graphics Forum (Proc. of Pacific Graphics 2008)*, 27(7), 2008.

[20] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics Hardware*, pages 53–60, 2006.

[21] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 24(3):1098–1107, 2005.

[22] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, and Hujun Bao. An efficient gpu-based approach for interactive global illumination. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 28(3):1–8, 2009.

[23] Gregory J. Ward and Paul S. Heckbert. Irradiance gradients. In *Eurographics Workshop on Rendering*, pages 85–98, 1992.

[24] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*, 27(5):1–11, 2008.

Analysis and design of the dynamical stability of collective behavior in crowds

Albert Mukovskiy
Section for Computational Sensomotorics, Department of Cognitive Neurology, Hertie Institute for Clinical Brain Research & Centre for Integrative Neuroscience, University Clinic, Tübingen, Germany
albert.mukovskiy@medizin.uni-tuebingen.de

Jean-Jacques E. Slotine
Nonlinear Systems Laboratory, Department of Mechanical Engineering, MIT; Cambridge, MA, USA
jjs@mit.edu

Martin A. Giese
Section for Computational Sensomotorics, Department of Cognitive Neurology, Hertie Institute for Clinical Brain Research & Centre for Integrative Neuroscience, University Clinic, Tübingen, Germany
martin.giese@uni-tuebingen.de

ABSTRACT

The modeling of the dynamics of the collective behavior of multiple characters is a key problem in crowd animation. Collective behavior can be described by the solutions of large-scale nonlinear dynamical systems that describe the dynamical interaction of locomoting characters with highly nonlinear articulation dynamics. The design of the stability properties of such complex multi-component systems has been rarely studied in computer animation. We present an approach for the solution of this problem that is based on Contraction Theory, a novel framework for the analysis of the stability complex nonlinear dynamical systems. Using a learning-based realtime-capable architecture for the animation of crowds, we demonstrate the application of this novel approach for the stability design for the groups of characters that interact in various ways. The underlying dynamics specifies control rules for propagation speed and direction, and for the synchronization of the gait phases. Contraction theory is not only suitable for the derivation of conditions that guarantee global asymptotic stability, but also of minimal convergence rates. Such bounds permit to guarantee the temporal constraints for the order formation in self-organizing interactive crowds.

Keywords: computer animation, crowd animation, coordination, distributed control, stability.

1 INTRODUCTION

Dynamical systems are frequently applied in crowd animation for the simulation of autonomous and collective behavior of many characters [MT01], [TCP06]. Some of this work has been inspired by observations in biology, showing that coordinated behavior of large groups of agents, such as flocks of birds, can be modelled as emergent behavior that arises from the dynamical coupling between interacting agents, without requiring an external central mechanism that ensures coordination [CS07, Cou09], [CDF+01]. Such models can be analyzed by application of methods from nonlinear dynamics [PRK03]. The simulation of collective behavior by self-organization in systems of dynamically coupled agents is interesting because it might reduce the computational costs of traditional computer animation techniques, such as scripting or path planning [TCP06, Rey87]. In addition, the generation of collective behavior by self-organization allows to implement spontaneous adaptation to external perturbations or changes in the system architecture, such as the variation of the number of characters. However, due to the complexity of the models describing individual characters the mathematical analysis of the underlying dynamical systems is typically quite complicated.

In crowd animation, some recent studies have tried to learn interaction rules from the behavior of real human crowds [DH03], [PPS07], [LFCCO09]. Other work has tried to optimize interaction behavior in crowds by exhaustive search of the parameter space exploiting computer simulations by definition of appropriate cost functions (e.g. [HMFB01]). However, most of the existing approaches for the control of group motion in computer graphics have not taken into account the effects of the articulation during locomotion on the control dynamics [PAB07], [NGCL09], [KLLT08]. Consequently, the convergence and stability properties of such dynamical animation systems have rarely been addressed. Distributed control theory has started to study the temporal and spatial self-organization of crowds of agents, and the design of appropriate dynamic interactions, typically assuming rather simple and often even linear agent models (e.g. [SS06], [PLS+07], [SDE95]). However, human-like characters are characterized by highly complex kinematic and even dynamic properties, c.f. [BH97]. Consequently, approaches for a systematic analysis and design of the dynamical properties

of crowd animation systems are largely lacking. However, such methods seem highly desirable, since they permit one to guarantee desired system properties and to ensure the robustness of the generated behavior under variations of system inputs and the system parameters.

In this paper we introduce Contraction Theory ([LS98], [PS07]) as a framework that makes such stability problems tractable, even for characters with multiple coupled levels of control. Contraction Theory provides a useful tool specifically for modularity-based stability analysis and design [Slo03], [WS05]. This framework is applied to a simple learning-based animation architecture for the real-time synthesis of the movements of interacting characters, which is based on a method that approximates complex human behavior by relatively simple nonlinear dynamical systems [GMP+09], [PMSA09]. Consistent with related approaches in robotics [RI06], [BC89], [GRIL08], [Ijs08], [BRI06], [GTH98], [CS93], this method generates complex movements by the combination of the learned movement primitives [OG06], [GMP+09]. The resulting system architecture is rather simple, making it suitable for a mathematical treatment of dynamical stability properties.

The paper is structured as follows: The structure of the animation system is sketched in section 2. The dynamics underlying navigation control is described in section 3. Subsequently, in section 4 we introduce some basic ideas from Contraction Theory. The major results of our stability analysis and some demos of their applications to the control of crowds are described in section 5, followed by the conclusions.

2 SYSTEM ARCHITECTURE

Our investigation of the collective dynamics of crowds was based on a learning-based animation system, described in details in [GMP+09] (see Fig. 1). By applying anechoic demixing [OG06] to motion capture data, we learned spatio-temporal components. These source components were generated online by nonlinear dynamical systems, Andronov-Hopf oscillators. The mappings σ_j between the stable solutions of the nonlinear oscillators and the required source functions were learned by application of kernel methods [GMP+09]. Each character is modelled by a single limit cycle oscillator, whose solution is mapped by support vector regression (SVR) onto three source signals. These signals were then superimposed with different linear weights w_{ij} and phase delays τ_{ij} in order to generate the joint angle trajectories $\xi_i(t)$ (see Fig. 1). By blending of the mixing weights and the phase delays, intermediate gait styles were generated. This allowed us to simulate specifically walking along paths with different curvatures, changes in step length and walking style. Interactive behavior of multiple agents can be modelled by making the states of the oscillators and the mixing

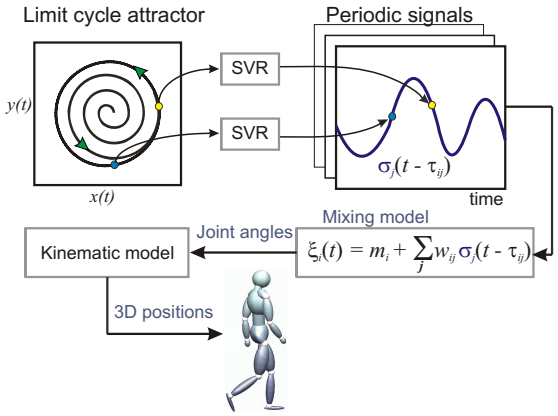


Figure 1: Architecture of the simulation system.

weights dependent on the behavior of the other agents. Such couplings result in a highly nonlinear system dynamics.

The heading direction of the characters was changed by morphing between curved gaits, controlled by a nonlinear navigation dynamics. In the shown applications this dynamics steers the avatars towards goal points that were placed along parallel straight lines. The heading dynamics was given by a nonlinear first-order differential equation (see [GMP+09] for details). Control of heading direction was only active during the the initial stage of the organization of the crowd, resulting in an alignment of the avatars along the parallel straight lines, independent of their initial positions and gait phases. (See Fig. 2 and Fig. 3).

3 CONTROL DYNAMICS

Beyond the control of heading direction, the analyzed scenarios of order formation in a group of characters require the control of the following variables: 1) phase within the step cycle, 2) step length, 3) gait frequency, and 4) heading direction.

The dynamics of each individual character was modelled by an Andronov-Hopf oscillator with constant equilibrium amplitude ($r_i^* = 1$). For appropriate choice of parameters, these nonlinear oscillators have a stable limit cycle that corresponds to a circular trajectory in phase space [AVK87].

In polar coordinates and with the instantaneous eigenfrequency ω this dynamics is given by: $\dot{r}(t) = r(t)\left(1 - r^2(t)\right)$, $\dot{\phi}(t) = \omega$. Control affects the instantaneous eigenfrequency ω of the Andronov-Hopf oscillators and their phases ϕ, while the first equation guarantees that the state stays on the limit cycle ($r(t) = 1, \forall t$).

The position z_i of each character along the parallel paths (see Fig. 2) fulfills the differential equation $\dot{z}_i(t) = \dot{\phi}_i g(\phi_i)$, where the positive function g determines the propagation speed of the character depending on the phase within the gait cycle. This nonlinear

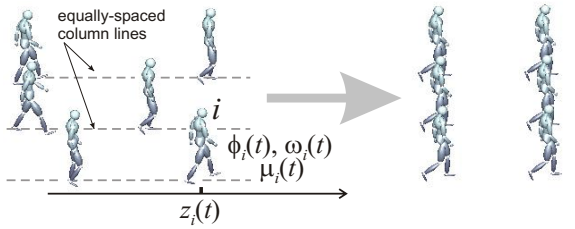


Figure 2: Crowd coordination scenario. Every character i is characterized by its position $z_i(t)$, the phase $\phi_i(t)$ and the instantaneous eigenfrequency $\omega_i(t) = \dot{\phi}_i(t)$ of the corresponding Andronov-Hopf oscillator, and a step-size scaling parameter $\mu_i(t)$.

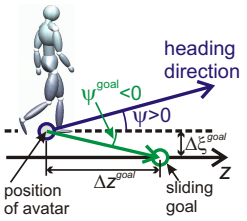


Figure 3: A sliding goal for each avatar was placed on a straight line at fixed distance ahead in z-direction. Heading direction angle: $\psi^{heading}$ and goal direction angle: ψ^{goal}.

function was determined empirically from a kinematic model of character. By integration of this propagation dynamics one obtains $z_i(t) = G(\phi_i(t) + \phi_i^0) + c_i$, with an initial phase shift ϕ_i^0 and some constant c_i depending on the initial position and phase of avatar i, and with the monotonously increasing function $G(\phi_i) = \int_0^{\phi_i} g(\phi) d\phi$, assuming $G(0) = 0$. Three control rules described:

I) Control of step frequency: A simple form of speed control is based on making the frequency of the oscillators $\dot{\phi}_i$ dependent on the behavior of the other characters. Let ω_0 be the equilibrium frequency of the oscillators without interaction. Then a simple controller is defined by the differential equation

$$\dot{\phi}_i(t) = \omega_0 - m_d \sum_{j=1}^{N} K_{ij}[z_i(t) - z_j(t) - d_{ij}] \qquad (1)$$

The constants d_{ij} specify the stable pairwise relative distances in the formed order for each pair (i, j) of characters. The elements of the link adjacency matrix $\mathbf{K}$ are $K_{ij} = 1$ if characters i and j are coupled and zero otherwise. In addition, we assume $K_{ii} = 0$. The constant $m_d > 0$ defines the coupling strength.

With the Laplacian $\mathbf{L}^d$ of the coupling graph, which is defined by $L_{ij}^d = -K_{ij}$ for $i \neq j$ and $L_{ii}^d = \sum_{j=1}^{N} K_{ij}$, and the constants $c_i = -\sum_{j=1}^{N} K_{ij} d_{ij}$ the last equation system can be re-written in vector form:

$$\dot{\phi} = \omega_0 \mathbf{1} - m_d(\mathbf{L}^d G(\phi + \phi^0) + \mathbf{c}) \qquad (2)$$

II) Control of step length: Step length was varied by morphing between gaits with short and long steps. A

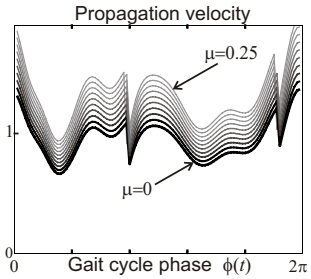


Figure 4: Propagation velocity for 10 different values the of step length morphing parameter $\mu = [0 \ldots 0.25]$ dependent on gait cycle phase $\phi(t)$ and $\omega(t) = 1$. The vertical axis is scaled in order to make all average velocities equal to one for $\mu = 0$ (lowest thick line). This empirical estimates are well approximated by $(1+\mu)g(\phi(t))$.

detailed analysis shows that the influence of step length on the propagation could be well captured by simple linear rescaling. If the propagation velocity of characters i is $v_i(t) = \dot{z}_i(t) = \dot{\phi}_i(t) g(\phi_i(t)) = \omega_i(t) g(\phi_i(t))$ for the normal step size, then the velocity for modified step size was well approximated by $v_i(t) = \dot{z}_i(t) = (1+\mu_i)\omega_i(t) g(\phi_i(t))$ with the morphing parameter μ_i. The range of morphing parameters was restricted to the interval $-0.5 < \mu_i < 0.5$, where this linear scaling law was fulfilled with high accuracy. The empirically estimated propagation velocity in heading direction, dependent on gait phase, is shown in Fig.4 for different values of the step length morphing parameter μ_i. Using the same notations as in equation (1), this motivates the definition of the following dynamics that models the influence of the step length control on the propagation speed:

$$\dot{\mathbf{z}} = \omega g(\phi + \phi^0)(1 - m_z(\mathbf{L}^z \mathbf{z} + \mathbf{c})) \qquad (3)$$

In this equation $\mathbf{L}^z$ signifies the Laplacian of the relevant coupling graph, and m_z the strength of the coupling. For uncoupled characters ($m_z = 0$) this equation is consistent with the the definition of propagation speed that was given before.

III) Control of step phase: By defining separate controls for step length and step frequency it becomes possible to dissociate the control of position and step phase of the characters. Specifically, it is interesting to introduce a controller that results in phase synchronization between different characters. This can be achieved by addition of a simple linear coupling term to equation (1), written in vector form:

$$\dot{\phi} = \omega_0 \mathbf{1} - m_d(\mathbf{L}^d G(\phi + \phi^0) + \mathbf{c}) - k\mathbf{L}^\phi \phi \qquad (4)$$

with $k > 0$ and the Laplacian $\mathbf{L}^\phi$. (All sums or differences of angular variables were computed by modulo 2π).

IV) Control of heading direction:

The heading directions of the characters were controlled by a navigation dynamics that steers the avatars

towards goal points, which were placed along parallel straight lines in front of the avatars (2). The heading dynamics was given by a nonlinear differential equation, independently for every character [GMP+09]:

$$\dot{\psi}_i = \sin(\psi_i^{goal} - \psi_i) \tag{5}$$

where $\psi_i^{goal} = \arctan(\Delta\xi_i^{goal}/\Delta z_i^{goal})$, $\Delta\xi_i^{goal}$ is the distance to the goal line in the direction orthogonal to the propagation direction, while Δz_i^{goal} is constant, (see Fig. 3). The morphing weight that controls the mixture of walking with left and right turning was proportional to $\dot{\psi}_i(t)$. For the mathematical stability analysis presented in the following, we neglected the influence of the dynamics of the control of heading direction, focusing on the order formation scenarios when the agents' heading directions are already aligned, when they walk along parallel straight lines towards sliding goal points. In this case, the positions of the agents can be described by a single position variable $z(t)$. An extension of the developed analysis framework including the control of the heading direction is in progress.

The mathematical results derived in the following sections apply to subsystems of the complete system dynamics that is given by equations (3) and (4). In addition, simulations are presented for the full system dynamics.

4 CONTRACTION THEORY

Dynamical systems describing the behavior of autonomous agents are essentially nonlinear. In contrast to the linear dynamical systems, a major difficulty of the analysis of stability properties of nonlinear is that the stability properties of parts usually do not transfer to composite systems. Contraction Theory [LS98] provides a general method for the analysis of essentially nonlinear systems, which permits such a transfer, making it suitable for the analysis of complex systems with many components. Contraction Theory characterizes the system stability by the behavior of the differences between solutions with different initial conditions. If these differences vanish exponentially over time, all solutions converge towards a single trajectory, independent from the initial states. In this case, the system is called *globally asymptotically stable*. For a general dynamical system of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \tag{6}$$

assume that $\mathbf{x}(t)$ is one solution of the system, and $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) + \delta\mathbf{x}(t)$ a neighboring one with a different initial condition. The function $\delta\mathbf{x}(t)$ is also called *virtual displacement*. With the Jacobian of the system $\mathbf{J}(\mathbf{x}, t) = \frac{\partial \mathbf{f}(\mathbf{x},t)}{\partial \mathbf{x}}$ it can be shown [LS98] that any nonzero virtual displacement decays exponentially to zero over time if the symmetric part of the Jacobian $\mathbf{J}_s = (\mathbf{J} + \mathbf{J}^T)/2$ is uniformly negative definite, denoted as $\mathbf{J}_s < 0$. This implies that it has only negative eigenvalues for all relevant state vectors $\mathbf{x}$. In this case, it can be shown that the norm of the virtual displacement decays at least exponentially to zero, for $t \to \infty$. If the virtual displacement is small enough, then

$$\dot{\delta\mathbf{x}}(t) = \mathbf{J}(\mathbf{x}, t)\delta\mathbf{x}(t)$$

implying through $\frac{d}{dt}||\delta\mathbf{x}(t)||^2 = 2\delta\mathbf{x}^T(t)\mathbf{J}_s(\mathbf{x}, t)\delta\mathbf{x}$ the inequality: $||\delta\mathbf{x}(t)|| \leq ||\delta\mathbf{x}(0)||\, e^{\int_0^t \lambda_{\max}(\mathbf{J}_s(\mathbf{x},s))\, ds}$. This implies that the virtual displacements decay with a *convergence rate* (inverse timescale) that is bounded from below by the quantity $\rho_c = -\sup_{\mathbf{x},t} \lambda_{\max}(\mathbf{J}_s(\mathbf{x}, t))$, where $\lambda_{\max}(.)$ signifies the largest eigenvalue. With $\rho_c > 0$ all trajectories converge to a single solution exponentially in time [LS98].

Contraction analysis can be applied also to hierarchically coupled systems [LS98]. Consider a composite dynamical system with two components, where the dynamics of the first subsystem is not influenced by the dynamics of the second one. Such system is called *hierarchically coupled*. The composite dynamical system:

$$\frac{d}{dt}\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}_1) \\ \mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2) \end{pmatrix} \tag{7}$$

results in the Jacobian:

$$\mathbf{F} = \begin{pmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x}_1)}{\partial \mathbf{x}_1} & 0 \\ \frac{\partial \mathbf{f}_2(\mathbf{x}_1,\mathbf{x}_2)}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{f}_2(\mathbf{x}_1,\mathbf{x}_2)}{\partial \mathbf{x}_2} \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{11} & 0 \\ \mathbf{F}_{21} & \mathbf{F}_{22} \end{pmatrix} \tag{8}$$

Consider then the smooth dynamics of virtual displacements: $\frac{d}{dt}\begin{pmatrix} \delta\mathbf{x}_1 \\ \delta\mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{11} & 0 \\ \mathbf{F}_{21} & \mathbf{F}_{22} \end{pmatrix}\begin{pmatrix} \delta\mathbf{x}_1 \\ \delta\mathbf{x}_2 \end{pmatrix}$, where $\mathbf{F}_{21}$ is bounded. The first subsystem does not depend on the second, so that $\delta\mathbf{x}_1$ exponentially converges to 0 if $(\mathbf{F}_{11})_s < 0$. Then, $\mathbf{F}_{21}\delta\mathbf{x}_1$ is an exponentially decaying disturbance for the second subsystem. In this case, (see [LS98] for details of proof), uniformly negative definite $\mathbf{F}_{22}$ implies exponential convergence of $\delta\mathbf{x}_2$ to an exponentially decaying ball. The whole system is then globally exponentially convergent to a single trajectory.

Many systems are not contracting with respect to all dimensions of the state space, but show convergence with respect to a subset of dimensions. Such behavior can be mathematically characterized by *partial contraction* [WS05], [PMSA09]. The underlying idea is to construct an auxiliary system that is contracting with respect to a subset of the arguments of the function $\mathbf{f}$. The major result is the following:

Theorem 1 *Consider a nonlinear system of the form*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{x}, t) \tag{9}$$

and assume the existence of auxiliary system

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{x}, t) \tag{10}$$

that is contracting with respect to $\mathbf{y}$ *uniformly for all relevant* $\mathbf{x}$*. If a particular solution of this auxiliary system verifies a specific smooth property, then trajectories of the original system (9) verify this property exponentially. The original system is then said to be partially contracting. [WS05].*

A 'smooth property' is a property of the solution that depends smoothly on space and time, such as convergence against a particular solution or a properly defined distance to a subspace in phase space. The proof of the theorem is immediate noticing that the observer-like system (10) has $\mathbf{y}(t) = \mathbf{x}(t)$ for all $t \geq 0$ as a particular solution. Since all trajectories of the $\mathbf{y}$-system converge exponentially to a single trajectory, this implies that also the trajectory $\mathbf{x}(t)$ verifies this specific property with exponential convergence.

It is thus sufficient to show that the auxiliary system is contracting in order to prove the convergence to a subspace. Let us assume that system has a flow-invariant *linear subspace* $\mathscr{M}$, which is defined by the property that trajectories starting in this space always remain in it for arbitrary times ($\forall t : \mathbf{f}(\mathscr{M},t) \subset \mathscr{M}$). If matrix $\mathbf{V}$ is an orthonormal projection onto $\mathscr{M}^{\perp}$, then sufficient condition for global exponential convergence to $\mathscr{M}$ is:

$$\mathbf{V}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{\mathrm{s}} \mathbf{V}^T < \mathbf{0}, \tag{11}$$

where smaller sign indicates that this matrix is negative definite (see [PS07, PMSA09]).

5 RESULTS

We derived contraction bounds for three scenarios that correspond to control dynamics with increasing levels of complexity.

1) Control of step phase without position control: The simplest case is a control of the phase within the step cycle of the walkers without simultaneous control of the position of the characters. Such simple control already permits to simulate interesting behaviors, such as soldiers synchronizing their step phases [PMSA09], [**Demo**[1]]. The underlying dynamics is given by (4) with $m_d = 0$. For N identical dynamical systems, with symmetric identical coupling gains k this dynamics can be written

$$\dot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}_i) + k \sum_{j \in \mathscr{N}_i} (\mathbf{x}_j - \mathbf{x}_i), \quad \forall i = 1,\ldots,N \tag{12}$$

where $\mathscr{N}_i$ defines the index set specifying the neighborhood in the coupling graph, i.e. the other subsystems or characters that are coupled with character i.

This type of symmetric coupling, where the interaction forces between subsystems depend only on the differences of the phase variables is called *diffusive coupling*. In this case, the *Laplacian matrix* of the coupling scheme is given by $\mathbf{L} = \mathbf{L}_G \bigotimes I_p$, where p is the dimensionality of the individual sub-systems, and where $\bigotimes$ signifies the *Kronecker product*. The Laplacian of the coupling graph is the matrix $\mathbf{L}_G$. The system then can be rewritten compactly as $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},t) - k\mathbf{L}\mathbf{x}$ with the concatenated phase variable $\mathbf{x} = [\mathbf{x}_1^T,...,\mathbf{x}_n^T]^T$. The Jacobian of this system is $\mathbf{J}(\mathbf{x},t) = \mathbf{D}(\mathbf{x},t) - k\mathbf{L}$, where the block-diagonal matrix $\mathbf{D}(\mathbf{x},t)$ has the Jacobians of the uncoupled components $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i,t)$ as entries.

The dynamics has a flow-invariant linear subspace $\mathscr{M}$ that contains the particular solution $\mathbf{x}_1^* = \cdots = \mathbf{x}_n^*$. For this solution all state variables $\mathbf{x}_i$ are identical and thus in synchrony. In addition, for this solution the coupling term in equation (12) vanishes, so that the form of the solution is identical with the solution of the uncoupled systems $\dot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}_i)$. If $\mathbf{V}$ is a projection matrix onto the subspace $\mathscr{M}^{\perp}$, then, according to (11), the sufficient contraction condition for convergence toward $\mathscr{M}$ is given by $\mathbf{V}(\mathbf{D}(\mathbf{x},t) - k\mathbf{L})_s\mathbf{V}^T < \mathbf{0}$, [PMSA09]. This implies

$$\lambda_{\min}\left(\mathbf{V}(k\mathbf{L})_s\mathbf{V}^T\right) = k\lambda_{\mathbf{L}}^{+} > \sup_{\mathbf{x},t} \lambda_{\max}\left(\mathbf{D}_s\right)$$

with $\lambda_{\mathbf{L}}^{+}$ being the smallest non-zero eigenvalue of symmetrical part of the Laplacian $\mathbf{L}_s$. The maximal eigenvalue for the individual oscillator is $\sup_{\mathbf{x},t} \lambda_{\max}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x},t)\right)$. The sufficient condition for global stability of the overall system is given by $k > \sup_{\mathbf{x},t} \lambda_{\max}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x},t)\right)/\lambda_{\mathbf{L}}^{+}$. This implies the following minimum convergence rate: $\rho_{\mathrm{c}} = -\sup_{\mathbf{x},t} \lambda_{\max}(\mathbf{V}(\mathbf{D}(\mathbf{x},t) - \mathbf{L})_{\mathrm{s}}\mathbf{V}^T)$.

For the special case of (4) with $m_d = 0$ this implies the sufficient contraction conditions $k > 0$ and $(\mathbf{L}^{\phi})_s \geq 0$.

Different topologies of the coupling graphs result in different stability conditions, since for example $\lambda_{\mathbf{L}}^{+} = 2(1 - \cos(2\pi/N))$ for symmetric ring coupling, and $\lambda_{\mathbf{L}}^{+} = N$ for all-to-all coupling. (N is the number of avatars.) See [WS05] and [PMSA09] for details.

2) Speed control by variation of step frequency: The dynamics of this system is given by equations (2) and (3) for $m_z = 0$. Assuming arbitrary initial distances and phase offsets for different propagating characters, implying $G(\phi_i^0) = c_i$, $c_i \neq c_j$, for $i \neq j$, we redefine d_{ij} as $d_{ij} - (c_i - c_j)$ in (1), and accordingly redefine $\mathbf{c}$ in (2). Assuming this control dynamics, and two avatars i and j that follow a leading avatar, their phase trajectories converge to a single unique trajectory only if $c_i = c_j$. This is a consequence of the one-to-one correspondence between gait phase and position of the avatar that is given by equation (2). In all other cases the trajectories of the followers converge to one-dimensional, but distinct, attractors in phase-position space that are uniquely defined by c_i. These attractors correspond to a behavior where the follower's position oscillates around the position of the leader.

[1] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video0.avi

For the analysis of contraction properties we regard an auxiliary system obtained from (2) by keeping the terms which are only dependent on ϕ: $\dot{\phi} = -m_d \mathbf{L}^d G(\phi + \phi^0)$. The symmetrized Jacobian of this system projected to the orthogonal compliment of flow-invariant linear subspace $\phi_1^* + \phi_1^0 = \ldots = \phi_N^* + \phi_N^0$ determines whether this system is partially contracting. By virtue of a linear change of variables the study of the contraction properties of this system is equivalent to study the contraction properties of the dynamical system $\dot{\phi} = -m_d \mathbf{L}^d G(\phi)$ on trajectories converging towards its flow-invariant manifold, the linear subspace of $\phi_1^* = \ldots = \phi_N^*$.

In order to derive an asymptotic stability condition, we consider the following auxiliary system, corresponding to a part of (2): $\dot{\phi} = -m_d \mathbf{L}^d G(\phi)$. The Jacobian of this system is given by $\mathbf{J} = -m_d \mathbf{L}^d \mathbf{D}_g$, where $(\mathbf{D}_g)_{ii} = g(\phi_i) = G'(\phi_i) > 0$ is a strictly positive diagonal matrix. Exploiting diagonal stability theory [Per69], it is straightforward to demonstrate that the auxiliary system is globally asymptotically stable and its state converges to an attractor with $\phi_1^* = \ldots = \phi_N^*$ for any initial condition assuming $(\mathbf{L}^d)_s \geq 0$ and $m_d > 0$. The sufficient conditions for asymptotic stability are satisfied for all types of symmetric diffusive couplings with positive coupling strength. For the case of asymmetric coupling graphs with more general structure including negative feedback links some results on asymptotic stability have been provided in [SA08].

The sufficient conditions for (exponential) partial contraction towards flow-invariant subspace are, (see (11)): $\mathbf{V}\mathbf{J}_s(\phi)\mathbf{V}^T = -m_d \mathbf{V}\mathbf{B}(\phi)\mathbf{V}^T < 0$, introducing $\mathbf{B}(\phi) = \mathbf{L}^d\mathbf{D}_g + \mathbf{D}_g(\mathbf{L}^d)^T$ and $\mathbf{V}$ signifying the projection matrix onto the orthogonal complement of the flow-invariant linear subspace. For diffusive coupling with symmetric Laplacian the linear flow-invariant manifold $\phi_1^* = \ldots = \phi_N^*$ is also the null-space of the Laplacian. In this case, the eigenvectors of the Laplacian that correspond to positive eigenvalues can be used to construct the projection matrix $\mathbf{V}$. For $m_d > 0$ the contraction conditions are thus satisfied if $\mathbf{V}\mathbf{B}(\phi)\mathbf{V}^T = \mathbf{V}(\mathbf{L}^d\mathbf{D}_g + \mathbf{D}_g(\mathbf{L}^d)^T)\mathbf{V}^T > 0$ for any diagonal matrix $\mathbf{D}_g > 0$.

Next we prove the exponential contraction conditions for the particular case of symmetrical all-to-all coupling. In this case $\mathbf{L}^d = N\mathbf{I} - \mathbf{1}\mathbf{1}^T \geq 0$, where $\mathbf{I}$ is identity matrix of size N. Since $\mathbf{V}\mathbf{1} = \mathbf{1}^T\mathbf{V}^T = 0$, we obtain $\frac{1}{2}\mathbf{V}(\mathbf{L}^d\mathbf{D}_g + \mathbf{D}_g(\mathbf{L}^d)^T)\mathbf{V}^T = N\mathbf{V}(\mathbf{D}_g)\mathbf{V}^T > 0$ for $\mathbf{D}_g > 0$. A lower bound for the contraction rate is computed from the projected symmetrized Jacobian $\mathbf{V}\mathbf{J}_s(\phi)\mathbf{V}^T = -\frac{m_d}{2}\mathbf{V}\mathbf{B}(\phi)\mathbf{V}^T$. Contraction theory also permits to compute the guaranteed contraction rate $\rho_{min} = m_d \min_\phi (g(\phi))\lambda^+_{\mathbf{L}^d}$, with $\lambda^+_{\mathbf{L}^d} = N$ for all-to-all symmetric coupling.

For a general symmetric couplings with positive links (with equal coupling strength $m_d > 0$) we obtain the sufficient contraction condition as: $\lambda^+_{min}(\mathbf{L}^d)/\lambda^+_{max}(\mathbf{L}^d) > \max_\phi(|g(\phi) - mean(g(\phi))|)/mean(g(\phi))$, where mean value of $g(\phi)$ over the gait cycle period T is: $mean(g(\phi)) = 1/T \int_0^T g(\phi)d\phi$. This condition is derived from the fact that for symmetric (positive) matrices M_1 and M_2 for $M_1 - M_2 > 0$ it is sufficient to satisfy $M_1 > M_2$ (the last means $\lambda_{min}(M_1) > \lambda_{max}(M_2)$). This sufficient condition put the constraints on admissible topologies of the coupling scheme dependent on the smoothness of gait velocity function $g(\phi)$. Alternatively, it is possible to introduce low-pass filtering of the forward kinematics of walking characters in order to increase the smoothness of $g(\phi)$.

An illustration of these stability bounds if given by the [**Demo**[2]]; that shows convergent behavior of the characters when the contraction condition $m_d > 0, (\mathbf{L}^d)_s \geq 0$ is satisfied for all-to-all coupling. [**Demo**[3]] shows the divergent behavior of a group when this condition is violated when $m_d < 0$.

The same proof can be extended for **nonlinear control rules**. In this case the eigenfrequency is given by a nonlinear modification of the control rule in eq. (1), for character i coupled to character j as: $\omega_i = \omega_0 + m_d h(z_j - z_i + d_{ij})$, where the saturating nonlinear function h could be given, for example by $h(z) = 1/[1 + \exp(-\gamma z)]$ with $\gamma > 0$. This nonlinear function limits the range of admissible speeds for the controller. Using the same notations as above, the dynamics of a single follower that follows a leader at position $P(t)$ is given by: $\dot{\phi}(t) = \omega_0 + m_d h(P(t) - G(\phi(t)) + c)$. The Jacobian of this dynamics $\mathbf{J}_s = -m_d h' g(\phi) < 0$ is negative, which follows from $m_d > 0$, $g(\phi) > 0$ and taking into account $h'(z) = dh(z)/dz > 0, \forall z$, what guarantees contraction.

Again this dynamics can be extended for N avatars, resulting in the nonlinear differential equation system: $\dot{\phi}_i(t) = \omega_0 - m_d \sum_{j=1}^N K_{ij} h(G(\phi_i) - G(\phi_j) + d_{ij}), \forall i$. The Jacobian of the system is: $\mathbf{J}(\phi) = -m_d \mathbf{L}^d(\phi)\mathbf{D}_g$, where $\mathbf{L}^d_{ij}(\phi) = -K_{ij}h'(G(\phi_i) - G(\phi_j) + d_{ij})$, $\mathbf{L}^d_{ii}(\phi) = \sum_{j\neq i}^N K_{ij}h'(G(\phi_i) - G(\phi_j) + d_{ij})$, $d_{ii} = 0$, ($\mathbf{D}_g$ is defined as before). Furthermore, the even function $h'(z) > 0$ implies that the Laplacian $\mathbf{L}^d(\phi)$ is symmetric diagonally dominant and it stays positive semidefinite for any positive $K_{ij} > 0$, by Gershgorin's Theorem [HJ85]. This implies that the system is asymptotically stable, its solutions converging to an attractor. The analysis of exponential convergence requires further steps that exceed the scope of this paper.

3) Stepsize control combined with a control of step phase: The dynamics is given by equations (3) and (4) with $m_d = 0$. This dynamics defines a hier-

[2] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video1.avi
[3] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video2.avi

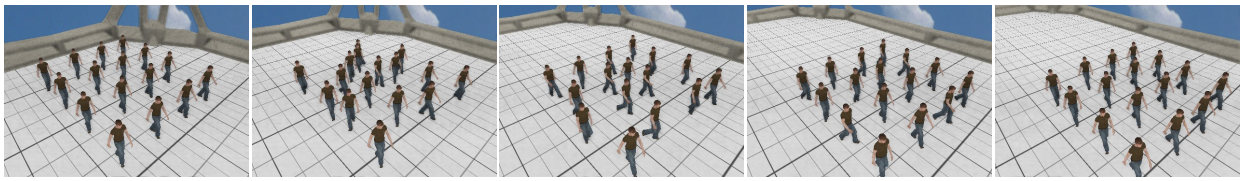


Figure 5: Self-organized reordering of a crowd with 16 characters. Control dynamics affects direction, distance and gait phase. See [**Demo**[7]].

archically coupled nonlinear system (see (7), Section 4), which is difficult to analyze with classical methods [LS98]. The dynamics for $\mathbf{z}(t)$ given by equation (3) is partially contracting in case of all-to-all coupling for any bounded external input $\phi(t)$, if $m_z > 0$, $\mathbf{L}^z \geq 0$ and $\omega(t) > 0$. These sufficient contraction conditions can be derived from the requirement of the positive-definiteness of the symmetrized Jacobian applying a similar technique as above. The Jacobian of this subsystem is $\mathbf{J}(\phi,\omega) = -m_z \mathbf{D}_g^z(\phi,\omega)\mathbf{L}^z$, with the diagonal matrix $(\mathbf{D}_g^z(\phi,\omega))_{ii} = \omega_i g(\phi_i + \phi_i^0) > 0$ that is positive definite since $g(\phi) > 0$ and $\omega > 0$. This subsystem is (exponentially) contracting and its relaxation rate is determined by $\rho_z = m_z \min_\phi (g(\phi))\lambda^+_{\mathbf{L}^z}$ (in the case of all-to-all coupling) for any input from the dynamics of $\phi(t)$ eq. (4). The last dynamics is contracting when $(\mathbf{L}^\phi)_s \geq 0$ and its relaxation rate is $\rho_\phi = k\lambda^+_{\mathbf{L}^\phi}$, where $\lambda^+_{\mathbf{L}^\phi}$ is the smallest non-zero eigenvalue of $(\mathbf{L}^\phi)_s$. The effective relaxation time of the overall dynamics is thus determined by the minimum of the contraction rates ρ_ϕ and ρ_z.

Demonstrations of this control dynamics satisfying the contraction conditions are shown in [**Demo**[4]], without control of step phase, and in [**Demo**[5]], with control of step phase.

4) Advanced scenarios: A simulation of a system with stable dynamics with both types of speed control (via step size and step frequency) and step phase control is shown in [**Demo**[6]]. Using the same dynamics, a larger crowd of 16 avatars simulated with the open-source animation engine Horde3d [Sch09] is shown in [**Demo**[7]]. In this scenario, dynamic obstacle avoidance and control of heading direction were activated in an initial time interval for unsorting of a formation of avatars. In a second time interval navigation is deactivated, and speed and position control according to the discussed principles takes over. [**Demo**[8]] demonstrates a large synchronizing crowd with 36 avatars without initial reordering. [**Demo**[9]] shows the divergence dynamics of the crowd from previous example, when negative distance-to-eigenfrequency coupling strength used ($m_d < 0$). Two videos [**Demo**[10]] and [**Demo**[11]] show convergence dynamics of the crowd of 49 avatars for two different values of the strength of the distance-to-step size coupling (slow and fast). The coupling strength for the phase coupling dynamics is constant for this example. The development of stability bounds and estimates of relaxation times for even more advanced scenarios including multiple control levels of navigation is the goal of ongoing work.

6 CONCLUSION

For the example of a learning-based system for the animation of locomoting groups, we have demonstrated first applications of Contraction Theory for the analysis and the design of stability and convergence properties of collective behaviors in animated crowds. The dynamics of the collective behavior of animated crowds is highly nonlinear and prevents the stability analysis using classical approaches. Opposed to these approaches, Contraction theory allows to transfer stability results from the components to composite systems. Future work has to extend this approach to more complex scenarios, especially including non-periodic movements.

ACKNOWLEDGEMENTS

Supported by DFG Forschergruppe 'Perceptual Graphics', the EC FP7/2007-2013 projects 'AMARSI' (grant agreement No.248311) and 'TANGO' and the Hermann und Lilly-Schilling-Stiftung. Authors thank Karsten Rohweder for help with animations in Horde3d.

REFERENCES

[AVK87] A. A. Andronov, A. A. Vitt, and S. E. Khaikin. *Theory of oscillators*. Dover Publ. Inc., New York, 1987.

[BC89] A. Bruderlin and T. W. Calvert. Goal-directed, dynamic animation of human walking. *Proc. of the 16th Conf. on Computer graphics and interactive techniques, ACM SIGGRAPH*, pages 233–242, 1989.

[BH97] D. C. Brogan and J. K. Hodgins. Group behaviors for systems with significant dynamics. *Autonomous Robots*, pages 137–153, 1997.

[4] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video3.avi
[5] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video4.avi
[6] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video5.avi
[7] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video6.avi
[8] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video7.avi
[9] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video8.avi

[10] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video9.avi
[11] www.uni-tuebingen.de/uni/knv/arl/avi/wscg/video10.avi

[BRI06] J. Buchli, L. Righetti, and A. J. Ijspeert. Engineering entrainment and adaptation in limit cycle systems - from biological inspiration to applications in robotics. *Biol. Cyb.*, 95, 6:645–664, 2006.

[CDF+01] S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, New Jersey, 2001.

[Cou09] I. D. Couzin. Collective cognition in animal groups. *Trends in Cogn. Sci.*, 13, 1:1–44, 2009.

[CS93] J. J. Collins and I. N. Stewart. Coupled nonlinear oscillators and the symmetries of animal gaits. *J. Nonlinear Sci.*, 3:349–392, 1993.

[CS07] F. Cucker and S. Smale. Emergent behavior in flocks. *IEEE Trans. Automat. Control*, 52, 5:852–862, 2007.

[DH03] W. Daamen and S. P. Hoogendoorn. Controlled experiments to derive walking behaviour. *European Journal of Transport and Infrastructure Research*, 3, 1:39–59, 2003.

[GMP+09] M. A. Giese, A. Mukovskiy, A. Park, L. Omlor, and J. J. E. Slotine. Real-time synthesis of body movements based on learned primitives. In D. Cremers et al., editor, *Stat. and Geom. Appr. to Vis. Mot. Anal., LNCS5604*, pages 107–127. Springer, 2009.

[GRIL08] A. Gams, L. Righetti, A. J. Ijspeert, and J. Lenarcic. A dynamical system for online learning of periodic movements of unknown waveform and frequency. *Proc. of the IEEE RAS / EMBS Int. Conf. on Biomedical Robotics and Biomechatronics*, pages 85–90, 2008.

[GTH98] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics based models. *Proc. ACM SIGGRAPH, Int. Conf. on Comp. Graph. and Interactive Techniques*, pages 9–20, 1998.

[HJ85] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1985.

[HMFB01] D. Helbing, P. Molnár, I. J. Farkas, and K. Bolay. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 28:361–383, 2001.

[Ijs08] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21, 4:642–653, 2008.

[KLLT08] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi. Group motion editing. *ACM Transactions on Graphics, SIGGRAPH 2008*, 27, 3:80–87, 2008.

[LFCCO09] A. Lerner, E. Fitusi, Y. Chrysanthou, and D. Cohen-Or. Fitting behaviors to pedestrian simulations. *Proc. Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 199–208, 2009.

[LS98] W. Lohmiller and J. J. E. Slotine. On contraction analysis for nonlinear systems. *Automatica*, 34, 6:683–696, 1998.

[MT01] S. R Musse and D. Thalmann. A behavioral model for real time simulation of virtual human crowds. *IEEE Trans. on Vis. and Comp. Graph.*, 7, 2:152–164, 2001.

[NGCL09] R. Narain, A. Golas, S. Curtis, and M. Lin. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics, Art.122*, 28, 5:1–8, 2009.

[OG06] L. Omlor and M. A. Giese. Blind source separation for over-determined delayed mixtures. *Adv. in NIPS*, 19:1049–1056, 2006.

[PAB07] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. *Proc. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 99–108, 2007.

[Per69] S. K. Persidskii. Problem of absolute stability. *Automat. Remote Control*, 12:1889–1895, 1969.

[PLS+07] D. A. Paley, N. E. Leonard, R. Sepulchre, D. Grunbaum, and J. K. Parrish. Oscillator models and collective motion: Spatial patterns in the dynamics of engineered and biological networks. *IEEE Control Systems Magazine*, 27:89–105, 2007.

[PMSA09] A. Park, A. Mukovskiy, J. J. E. Slotine, and Giese M. A. Design of dynamical stability properties in character animation. *Proc. of VRIPHYS 09*, pages 85–94, 2009.

[PPS07] S. Paris, J. Pettré, and Donikian S. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Proc. Eurographics 2007*, 26, 3:665–674, 2007.

[PRK03] A. Pikovsky, M. Rosenblum, and J. Kurths. *Synchronization, A Universal Concept in Nonlinear Sciences*. Cambridge University Press, Cambridge, 2003.

[PS07] Q. C. Pham and J. J. E. Slotine. Stable concurrent synchronization in dynamic system networks. *Neural Networks*, 20, 3:62–77, 2007.

[Rey87] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21, 4:25–34, 1987.

[RI06] L. Righetti and A. J. Ijspeert. Programmable central pattern generators: an application to biped locomotion control. *Proc. of the 2006 IEEE Int. Conf. on Rob. and Autom.*, pages 1585–1590, 2006.

[SA08] E. D. Sontag and M. Arcak. Passivity-based stability of interconnection structures. In V. et al. Blondel, editor, *Rec. Adv. in Learning and Control. Vol.371*, pages 195–204. Springer-Verlag, NY, 2008.

[Sch09] N. Schulz. http://www.horde3d.org/. 2006–2009.

[SDE95] G. Sch´oner, M. Dose, and C. Engels. Dynamics of behavior: Theory and applications for autonomous robot architectures. *Robotics and Autonomous Systems*, 16, 2-4:213–245, 1995.

[Slo03] J. J. E. Slotine. Modular stability tools for distributed computation and control. *Int. J. Adaptive Control and Signal Processing*, 17, 6:397–416, 2003.

[SS06] L. Scardovi and R. Sepulchre. Collective optimization over average quantities. *Proceedings of the 45th IEEE Conference on Decision and Control, San Diego, California*, pages 3369–3374, 2006.

[TCP06] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *Proc. ACM SIGGRAPH '06*, 25, 3:1160–1168, 2006.

[WS05] W. Wang and J. J. E. Slotine. On partial contraction analysis for coupled nonlinear oscillators. *Biological Cybernetics*, 92, 1:38–53, 2005.

Evaluation of the Linear Box-Spline Filter from Trilinear Texture Samples: A Feasibility Study

Balázs Domonkos
Mediso Medical Imaging Systems, Hungary
balazs.domonkos@mediso.hu

Balázs Csébfalvi
Budapest University of Technology and Economics
cseb@iit.bme.hu

ABSTRACT

The major preference for applying B-spline filtering rather than non-separable box spline filtering on the BCC lattice is the fact that separable filtering can be performed more efficiently on current GPUs due to the utilization of the hardware-accelerated trilinear texture fetching. In order to make a fair comparison, a similar, efficient evaluation scheme is required that uses trilinear texture fetches instead of nearest-neighbor ones also for the box splines. Thus, in this paper, we propose an evaluation scheme for the linear BCC box spline built upon a trilinear B-spline basis. We compare our trilinearly evaluated linear box spline scheme to the latest method, that uses twice as many nearest neighbor fetches. Then we give a comparison to the major competitive methods: the BCC B-spline filtering and the BCC DC-spline filtering in terms of their performance.

Keywords: Volume Rendering, Filtering, Reconstruction.

1 INTRODUCTION

In many applications in engineering and computing science, a continuous phenomenon is represented by its discrete samples. In order to operate on the underlying continuous function, first it has to be accurately reconstructed from its discrete representation. Reconstruction filters have received attention also in image processing and volume visualization since appropriate reconstruction of multivariate functions is a key step of the processing pipeline [2, 3, 17, 18].

According to the most commonly-used sampling scheme in practice, volumetric data is often acquired on a uniform lattice by regular sampling, while reconstruction is performed by convolution filtering. An appropriate choice of both the sampling lattice and the reconstruction filter kernel is of crucial importance as they together directly determine the quality of the reproduced continuous function and the efficiency of the reconstruction.

Recent results advocate the benefits of non-Cartesian lattices for regular sampling. The application of Body-Centered Cubic (BCC) sampling received increased attention from the perspective of continuous signal reconstruction in the last decade [5, 6, 11, 12]. This lattice is optimal for sampling 3D signals of isotropic bandwidth [19, 21], unlike the commonly used Cartesian Cubic (CC) lattice along with tensor-product reconstruction. To perfectly reconstruct a signal of a spherically bounded spectrum from its discrete representation, roughly 30% fewer samples per unit volume have to be taken on a BCC lattice than on an equivalent CC lattice. In addition to the improved spectral isotropy, this directly translates into an explicit reduction of the storage cost.

A crucial question of BCC sampling is the way in which the original continuous signal is reconstructed from its discrete samples. Although due to the shift-invariant property of the sampling lattice, the reconstruction can be implemented by a simple convolution, the choice of the filter kernel has a direct impact on both numerical accuracy and visual quality. Generally, an appropriate filter is chosen by making a compromise between quality and efficiency.

Currently, three promising resampling techniques exist for the BCC lattice that provide high visual quality, numerical accuracy, and efficiency at the same time: the box splines [11], the BCC B-splines [8, 6], and the BCC DC-splines [10]. As only the latter two methods can exploit the hardware-accelerated trilinear filtering, it has not been possible to make a fair comparison so far. To remedy this problem, we propose an algorithm that uses trilinear fetches for the box spline filtering as well. Since these filters have already been compared in terms of visual quality and numerical accuracy [6, 10, 13], in this paper, we focus on a fair comparison of their performance.

2 RELATED WORK

One of the most important aspects of rendering sampled data is how to perform proper and efficient resampling depending on the applied lattice. For the CC lattice, reconstruction filters are usually designed in 1D, and then

Plzen, Czech Republic.

extended to the trivariate setting by a separable tensor-product extension. However, the BCC lattice is not separable itself, therefore the advantageous properties of a 1D filter are not necessarily inherited in 3D by a separable extension [21, 22, 15].

The first reconstruction filters tailored to the geometry of the non-Cartesian lattices were proposed by Entezari et al. [11]. They applied *box splines*, that offer a mathematically elegant toolbox for constructing a class of multidimensional elements with flexible shape and support. Box splines are often considered as a generalization of B-splines to multivariate setting. Theoretically, the computational complexity of a box spline is lower than that of an equivalent B-spline, since its support is more compact and its total polynomial degree is lower. To investigate this potential also in practice, several attempts were made. Although de Boor's recurrence relation [9] is the most commonly used technique for evaluating box splines at an arbitrary position, it is computationally inefficient and has numerical instabilities [14]. Addressing this issue, Entezari et al. [12] derived a piecewise-polynomial representation of the linear and quintic box splines for the BCC lattice. In a CPU-based implementation, due to the smaller support of the box spline kernels, the data access cost of discrete BCC samples turned out to be twice as low as for the equivalent B-spline filters on the CC lattice [12]. Following their work, Finkbeiner et al. proposed an algorithm to convolve the BCC samples with these box spline kernels [13]. Though they applied early selection of polynomial segments of the piecewise polynomial form that enabled them to avoid a full kernel evaluation for each affected sample point, the theoretical advantages of box splines could not be exploited on the GPUs, which are rather optimized for separable filtering.

Another family of non-separable filters is represented by the *Voronoi splines* [16] that inherit the geometry of a sampling lattice through its Voronoi cell. For Cartesian lattices, Voronoi splines coincide with tensor-product B-splines. For the 2D hexagonal lattice, Voronoi splines were originally proposed by Van de Ville et al. [23] as Hex-splines. For the BCC lattice Voronoi splines were derived as BCC-splines by Csébfalvi [5]. Recently, Mirzargar et al. [16] formulated the BCC-splines in terms of multi-box splines. In spite of their theoretical elegance, Voronoi splines are currently impractical, since their piecewise evaluation is not known yet.

Csébfalvi recommended a *prefiltered Gaussian reconstruction scheme* [4] adapting the principle of generalized interpolation [1] to the BCC lattice. According to this approach, first a non-separable discrete prefiltering is performed as a preprocessing step, and afterwards a fast separable Gaussian filtering is used for continuous resampling on the fly. This method was extended also to the *B-spline family of filters* [8]. An efficient GPU implementation was proposed exploiting the fact that the BCC lattice consists of two interleaved CC lattices, where the second CC lattice is translated by half of the grid spacing. The reconstruction can be performed separately for these two CC lattices in the given sample position by using a standard trilinear or tricubic B-spline resampling, and then the contributions are averaged [8]. BCC B-splines reconstruction was reported to be four to five times faster on an NVIDIA GeForce 6800 graphics card than a non-separable box spline reconstruction of the same approximation power [6], since the B-splines can utilize the hardware-accelerated trilinear texture fetching [20].

Recently, Domonkos et al. [10] proposed a *discrete/-continuous filter family* generated by the impulse response of the BCC trilinear kernel. This technique is theoretically equivalent to the discrete upsampling of the BCC-sampled volume on a higher resolution CC lattice, where the standard trilinear interpolation is used for resampling. In practice, however, the missing CC samples are calculated on the fly and not in a preprocessing. Using an optimized GPU implementation, the linear DC-spline was reported to be slightly faster than the linear box spline.

3 SPLINE RECONSTRUCTION FOR THE BCC LATTICE

In the following, we briefly review the main properties of the BCC lattice, as well as the box spline, B-spline, and DC-spline family of filters, as they are applied for reconstruction on the BCC lattice.

3.1 BCC Lattice

The BCC lattice Λ_{BCC} is a discrete subgroup of $\mathbb{R}^3$ generated by integer linear combinations of the following basis vectors:

$$\begin{aligned} \Xi_{BCC} &= [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3] = \frac{1}{2}\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \\ \Lambda_{BCC} &= \left\{ \Xi_{BCC}\mathbf{i} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\} \subset \mathbb{R}^3. \end{aligned}$$

Besides, the BCC lattice points are located on a CC lattice with an additional sample placed in the center of each cube. Thus, the BCC lattice can also be considered as two interleaved CC lattices Λ_{CC_A} and Λ_{CC_B}. By shifting the secondary CC lattice Λ_{CC_B} by half of the grid spacing, the vertices of the secondary CC lattice are moved to the centers of the primary CC cells:

$$\begin{aligned} \Lambda_{BCC} &= \Lambda_{CC_A} \cup \Lambda_{CC_B} \\ \Lambda_{CC_A} &= \left\{ \mathbf{i} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\} \\ \Lambda_{CC_B} &= \left\{ \mathbf{i} + \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\}. \end{aligned} \tag{1}$$

On the other hand, the BCC lattice can be obtained also from a dense CC lattice by keeping only the lattice points whose coordinates have identical parity:

$$\Lambda_{BCC} = \left\{ \frac{1}{2} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \quad : \; i \equiv j \equiv k \pmod 2 \atop i,j,k \in \mathbb{Z} \right\}. \quad (2)$$

3.2 Box Splines for the BCC Lattice

A box spline M_{Ξ} is the shadow of a unit-hypercube in $\mathbb{R}^n$ projected to $\mathbb{R}^s, s \leq n$ where the projection is characterized by $\Xi = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_n] \in \mathbb{R}^{s \times n}$, $\boldsymbol{\xi}_i \in \mathbb{R}^s \backslash \mathbf{0}$ [9]. The shape, the continuity order, and the approximation power of a given box spline M_{Ξ} is determined by Ξ. The simplest box spline is constructed when $s = n$ as a normalized characteristic function of its support:

$$M_{\Xi}(\mathbf{x}) = \begin{cases} \frac{1}{\det \Xi} & \text{if } \Xi^{-1}\mathbf{x} \in [0,1)^n \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

When adding a further direction vector $\boldsymbol{\xi} \in \mathbb{R}^s$ to Ξ, $s < n$, the box spline $M_{[\Xi,\boldsymbol{\xi}]}$ is given by the convolution:

$$M_{[\Xi,\boldsymbol{\xi}]}(\mathbf{x}) = \int_0^1 M_{\Xi}(\mathbf{x} - t\boldsymbol{\xi})\, dt. \quad (4)$$

The linear box spline $M_{\Xi^1_{BCC}} \in C^0$ for the BCC lattice is constructed as a 3D shadow of a tesseract along its antipodal axis, resulting a function with a rhombic dodecahedron support, which is the first neighbors cell of the BCC lattice [12]:

$$\Xi^1_{BCC} = \begin{bmatrix} & & 1/2 \\ \Xi_{BCC} & & 1/2 \\ & & 1/2 \end{bmatrix}. \quad (5)$$

$M_{\Xi^1_{BCC}}$ has its maximum value at the center, and has a linear falloff towards the 14 first-neighbor vertices:

$$M_{\Xi^1_{BCC}}(\mathbf{x}) = \max(1 - x - y,\, 0), \quad (6)$$

where x is the largest and y is the second largest component of the absolute coordinates of $\mathbf{x}$ [12].

3.3 B-Splines for the BCC Lattice

The B-spline of order zero is defined as a box filter:

$$\beta^0(t) = \begin{cases} 1 & \text{if } |t| < \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Generally, the B-spline filter of order n is derived by successively convolving $\beta^0(t)$ n times with itself. The first-order B-spline is the linear interpolation filter or tent filter:

$$\beta^1(t) = \beta^0(t) * \beta^0(t) = \begin{cases} 1 - |t| & \text{if } |t| \leq 1 \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

The 1D B-splines can be extended to the 3D CC lattice by a tensor product extension. BCC B-spline resampling exploits the decomposition property of the BCC lattice (Eq. 1). The reconstruction is performed separately for the two CC sub-lattices in the given sample position by using a standard separable CC B-spline resampling, and then the contributions are simply averaged [8, 6]. This evaluation is equivalent to the convolution of the BCC samples with a B-spline kernel.

3.4 DC-Splines for the BCC Lattice

The BCC lattice can be obtained from a CC lattice by removing the lattice points whose coordinates have different parity (Eq. 2). The BCC trilinear interpolation reproduces these "missing CC samples" by interpolating between the available BCC samples on the fly using a discrete filter. The resultant impulse response χ^1_{BCC} of the linear BCC DC-spline is obtained by convolving this discrete filter with a scaled trilinear kernel $\beta^1(2\mathbf{x})$:

$$\chi^1_{BCC}(\mathbf{x}) = \beta^1(2\mathbf{x}) + \frac{1}{2}\sum_{k=1}^{6} \beta^1(2(\mathbf{x} - \nu_{\mathbf{k}})) \quad (9)$$

$$[\nu_{\mathbf{1}\ldots\mathbf{6}}] = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

4 EVALUATION OF THE LINEAR BOX SPLINE FROM TRILINEAR TEXTURE SAMPLES

The major preference for applying the BCC B-spline filtering over the non-separable box spline filtering is the fact that separable filtering can be performed significantly faster on current GPUs due to the utilization of the hardware-accelerated trilinear texture fetching [20].

In order to make a fair comparison, an efficient evaluation scheme is required that uses trilinear texture fetches instead of nearest neighbor ones also for the box splines. In the following, we propose an algorithm for evaluation of the linear BCC box spline built upon a trilinear B-spline basis.

According to Eq. 6, the support of $M_{\Xi^1_{BCC}}$ covers four BCC samples that form a tetrahedron, thus the B-form of resampling is [11]:

$$f(\mathbf{r}) = \sum_{i=1}^{4} s(\mathbf{r}_i) M_{\Xi^1_{BCC}}(\mathbf{r_i} - \mathbf{r}), \quad (10)$$

where $\mathbf{r}$ is an arbitrary resampling point and s is a 3D array of the discrete BCC samples. Direct implementation of this B-form is rather inefficient, since a full kernel evaluation is performed for each $\mathbf{r_i}$ sample point [13].

A more efficient piecewise-polynomial evaluation scheme can be set up, since it is possible to evaluate the ordering of the absolute coordinates of $\mathbf{r_i} - \mathbf{r}$ in advance for each $\mathbf{r_i}$ lattice points [13].

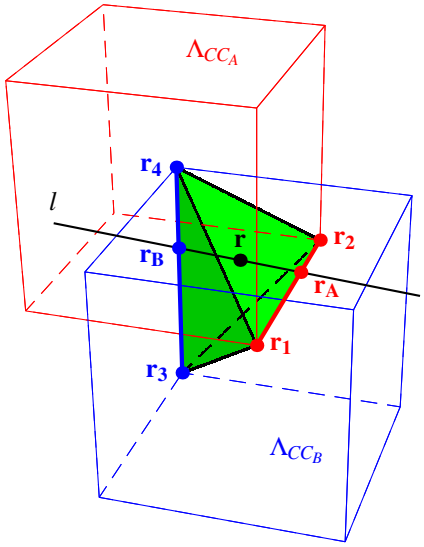


Figure 1: Trilinear evaluation scheme. For an arbitrary point $\mathbf{r}$, interpolation is performed within the green tetrahedron formed by the nearest points $\mathbf{r_1}, \mathbf{r_2} \in \Lambda_{CC_A}$ of the red CC lattice and the nearest points $\mathbf{r_3}, \mathbf{r_4} \in \Lambda_{CC_B}$ of the blue CC lattice. When $\mathbf{r}$ is an internal point, that is, $\mathbf{r} \notin \mathbf{r_{1,2}}$ and $\mathbf{r} \notin \mathbf{r_{3,4}}$, there is exactly one line l that intersects $\mathbf{r}$, and edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$.

4.1 Trilinear Evaluation Scheme

The key point of the derivation lies in the fact that the linear box spline constitutes a linear interpolator on the BCC lattice [11]. This enables us to evaluate the linear interpolation within the tetrahedron more efficiently than a direct evaluation of Eq. 10.

The first observation we make is that the tetrahedron is composed of four congruent isosceles triangles (see Fig. 1):

1. $\mathbf{r_{1,2,3}}$ 2. $\mathbf{r_{1,2,4}}$ 3. $\mathbf{r_{3,4,1}}$ 4. $\mathbf{r_{3,4,2}}$

Four edges of the tetrahedron are formed by the equal sides of these triangles with the length of $\frac{\sqrt{3}}{2}$ while the remaining two edges of the tetrahedron are formed by the sides $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ of the triangles with the length of 1:

$$\begin{aligned} \frac{\sqrt{3}}{2} &= |\mathbf{r_{1,3}}| = |\mathbf{r_{2,3}}| = |\mathbf{r_{1,4}}| = |\mathbf{r_{2,4}}| \\ 1 &= |\mathbf{r_{1,2}}| = |\mathbf{r_{3,4}}| \end{aligned}$$

The edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ overlap the edges of the BCC lattice. Moreover, when the BCC lattice is considered as two interleaved CC lattices (Eq. 1), edge $\mathbf{r_{1,2}}$ is contained by the first CC lattice Λ_{CC_A}, while edge $\mathbf{r_{3,4}}$ is contained by the second CC lattice Λ_{CC_B}.

This enables us to rewrite the tetrahedral interpolation as the compound of three linear interpolations using the following scheme:

1. First, we define line l that contains $\mathbf{r}$ and intersects both $\mathbf{r_{1,2}} \in \Lambda_{CC_A}$ and $\mathbf{r_{3,4}} \in \Lambda_{CC_B}$ (see Fig. 1). The intersection points with edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ are $\mathbf{r_A}$ and $\mathbf{r_B}$, respectively. This decouples the BCC resampling problem into resamplings of two separate CC lattices, to Λ_{CC_A} and Λ_{CC_B}.

2. Next, the discrete data is resampled in $\mathbf{r_A}$ for Λ_{CC_A} and $\mathbf{r_B}$ for Λ_{CC_B} using a simple linear kernel:

$$\begin{aligned} f_A &= s_A(\mathbf{r_1} + |\mathbf{r_{1,A}}|\mathbf{r_{1,2}}) \\ f_B &= s_B(\mathbf{r_3} + |\mathbf{r_{3,B}}|\mathbf{r_{3,4}}), \end{aligned} \tag{11}$$

   where s_A and s_B are linearly addressable 3D arrays of the discrete CC samples corresponding to Λ_{CC_A} and Λ_{CC_B}, respectively.

3. Finally, the linear combination of the two CC samples is calculated:

$$f(\mathbf{r}) = f_A + \frac{|\mathbf{r} - \mathbf{r_A}|}{|\mathbf{r_{A,B}}|}(f_B - f_A). \tag{12}$$

The clear advantage of this evaluation scheme is that Step 2 can be performed by only two trilinear fetches on the GPU instead of four nearest neighbor fetches. Actually, these trilinear fetches involve in fact only 1D linear interpolations since $\mathbf{r_A}$ and $\mathbf{r_B}$ lie on a lattice edge. Regarding the storage scheme, the consequence is that the BCC samples need to be stored in two separate CC lattices, i.e. conventional 3D textures, to be able to exploit the trilinear fetching capability of the GPU just like in case of the BCC B-spline and the BCC DC-spline.

4.2 Orientation Cases

Addressing $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$, and $\mathbf{r_4}$ for an arbitrary $\mathbf{r}$ is required in Step 1 which needs some further explanation. Let $\mathbf{r_{base}} = \text{round}(\mathbf{r})$ be the nearest lattice point in Λ_{CC_A} and let $\mathbf{d} = \mathbf{r} - \mathbf{r_{base}}$ be the relative resampling coordinates with their absolute values $\mathbf{a} = [|d_x|, |d_y|, |d_z|]^T \in [0, \frac{1}{2})^3$ and their signs $\mathbf{s} = [\text{sgn}(d_x), \text{sgn}(d_y), \text{sgn}(d_z)]^T$. Considering the symmetries of the rhombic dodecahedral support of $M_{\Xi^1_{BCC}}$, six different orientations of the resampling tetrahedron can be distinguished (see Fig. 2). These six cases are the 3! possible orderings of the absolute coordinates $\mathbf{a}$ in Eq. 6 as it was reported in [13].

Since using any control flow statement in the resampling implementation dramatically cuts the performance of the GPUs which have a SIMD architecture, it is advisable to avoid this six-fold branching. Descending order of three scalars can be calculated in a SIMD-aware manner as:

$$\begin{aligned} x &= max(a_x, a_y, a_z) \qquad z = min(a_x, a_y, a_z) \\ y &= a_x + a_y + a_z - x - z. \end{aligned} \tag{13}$$

On the other hand, based on the sort order of a_x, a_y, and a_z all the six orientations of the resampling tetrahedron can be transformed back to the first one (the green

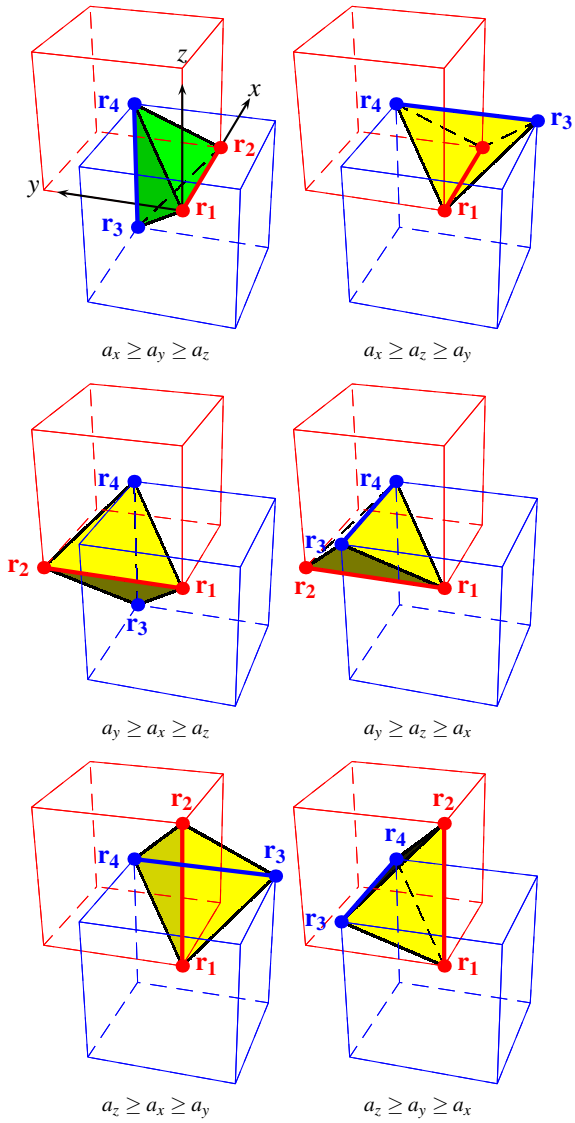


Figure 2: There are six orientation cases for ordering the coordinates of $\mathbf{a} \in [0, \frac{1}{2})^3$. The required resampling points $\mathbf{r_1}, \mathbf{r_2} \in \Lambda_{CC_A}$ and $\mathbf{r_3}, \mathbf{r_4} \in \Lambda_{CC_B}$ are determined by these six cases. These resampling points are indicated as red and blue dots for each orientation case.

tetrahedron for $a_x \geq a_y \geq a_z$ in Fig. 2). Thus, the resampling formula needs to be written only for the first orientation case, and the other cases can be retrieved by using this transformation. The transformation can be defined by a rotation matrix $\mathbf{\Pi}$ as

$$\Pi_{i,j} = s_i \cdot e_{\pi(j),i}, \tag{14}$$

where $\mathbf{e}_{\pi(1)}$, $\mathbf{e}_{\pi(2)}$, and $\mathbf{e}_{\pi(3)}$ are the unit vectors corresponding to x, y, and z, respectively (Eq. 13). As a compact notation, π represents the descending order of a_x, a_y, and a_z as a permutation. By using $\mathbf{\Pi}$, the lattice points can be addressed as

$$\mathbf{r_1} = \mathbf{r_{base}} + \mathbf{\Pi}\,[0\;0\;0]^T \qquad \mathbf{r_2} = \mathbf{r_{base}} + \mathbf{\Pi}\,[1\;0\;0]^T$$
$$\mathbf{r_3} = \mathbf{r_{base}} + \mathbf{\Pi}\left[\frac{1}{2}\;\frac{1}{2}\;-\frac{1}{2}\right]^T \qquad \mathbf{r_4} = \mathbf{r_{base}} + \mathbf{\Pi}\left[\frac{1}{2}\;\frac{1}{2}\;\frac{1}{2}\right]^T.$$

4.3 Formal Derivation

In the following, we also give a formal derivation of the proposed algorithm. The derivation is based on the rewriting of the tetrahedral interpolation in barycentric coordinates. Barycentric coordinates provide a convenient way for interpolation on a tetrahedral mesh:

$$f(\mathbf{r}) = \sum_{i=1}^{4} \lambda_i s(\mathbf{r_i}), \tag{15}$$

where scalars $\lambda_{1\ldots4}$ are barycentric coordinates of $\mathbf{r}$ with respect to the vertices of the tetrahedron $\mathbf{r_{1\ldots4}}$ under the constraint $\sum_{i=1}^{4} \lambda_i = 1$. The barycentric expansion of $\mathbf{r}$ is set up in terms of the vertices of the tetrahedron as:

$$\begin{aligned} \mathbf{T\lambda} &= \mathbf{r} - \mathbf{r_4} \\ \mathbf{T} &= [\mathbf{r_1} - \mathbf{r_4} \mid \mathbf{r_2} - \mathbf{r_4} \mid \mathbf{r_3} - \mathbf{r_4}] \\ \boldsymbol{\lambda} &= \left[\;\lambda_1 \quad \lambda_2 \quad \lambda_3\;\right]^T. \end{aligned} \tag{16}$$

The solution of this linear equation system is

$$\mathbf{T} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & -1 \end{bmatrix}, \quad \mathbf{T}^{-1} = \begin{bmatrix} -1 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix},$$

$$\lambda_1 = 1 - x - y \qquad \lambda_2 = x - y$$
$$\lambda_3 = y - z \qquad \lambda_4 = y + z.$$

This enables us to write Eq. 10 as

$$f(\mathbf{r}) = \sum_{i=1}^{2} \lambda_i s_A(\mathbf{r_i}) + \sum_{i=3}^{4} \lambda_i s_B(\mathbf{r_i}). \tag{17}$$

Using the separable trilinear technique of Sigg and Hadwiger [20], evaluation of Eq. 17 can be derived by two linear fetches instead of four nearest neighbor ones. In general, two nearest neighbor fetches can be replaced by a linear fetch as:

$$\begin{aligned} (1-t)\,f_i + t\,f_{i+1} &\;\Rightarrow\; f(i+t) \\ a\,f_i + b\,f_{i+1} &\;\Rightarrow\; (a+b)\,f\left(i + \frac{b}{a+b}\right), \end{aligned} \tag{18}$$

as long as $t \in [0,1]$ and $\frac{b}{a+b} \in [0,1]$. By combining both λ_1 with λ_2 and λ_3 with λ_4, the linear box spline can be evaluated by two linear texture fetches:

$$\sum_{i=1}^{2} \lambda_i s_A(\mathbf{r_i}) \;\Rightarrow\; (1-2y)\,s_A\left(\mathbf{r_1} + \frac{x-y}{1-2y}\mathbf{\Pi}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)$$
$$\sum_{i=3}^{4} \lambda_i s_B(\mathbf{r_i}) \;\Rightarrow\; 2y\,s_B\left(\mathbf{r_3} + \frac{y+z}{2y}\mathbf{\Pi}\begin{bmatrix}0\\0\\1\end{bmatrix}\right)$$

Summing these terms up, we get

$$
\begin{aligned}
f_A &= s_A\left(\mathbf{r_{base}} + \frac{x-y}{1-2y}\,\mathbf{s}\circ\mathbf{e}_{\pi(1)}\right) \qquad (19)\\
f_B &= s_B\left(\mathbf{r_{base}} + \mathbf{s}\circ\left(\begin{bmatrix}1/2\\1/2\\1/2\end{bmatrix} + \frac{z-y}{2y}\mathbf{e}_{\pi(3)}\right)\right),\\
f(\mathbf{r}) &= (1-2y)\,f_A + 2y\,f_B
\end{aligned}
$$

where $\circ$ represents the element-wise product. This is exactly what was claimed in Step 2 and Step 3 of the proposed evaluation scheme.

5 GPU IMPLEMENTATION

We employed the proposed trilinear evaluation scheme formulated in Eq. 19 in a GPU-based first-hit ray-casting application by using ray marching with equidistant steps. At each sample position, a filter kernel was used to reconstruct the volume from the discrete BCC samples. To get a numerically stable formulation when the resampling point lies within a triangular face, on an edge, or coincides a vertex, the divisions in Eq. 19 are evaluated as $\lim_{\varepsilon\to 0}\varepsilon\cdot s_i(\frac{\text{constant}}{\varepsilon}) = 0$. This numerical safeguard was incorporated in the GPU implementation as well.

In our GPU implementation, the lattice samples are stored as textures. Function $s_A(\mathbf{r})$ fetches the sample set s_A at $\mathbf{r}+[\frac{1}{2},\frac{1}{2},\frac{1}{2}]^T$, while function $s_B(\mathbf{r})$ fetches the shifted sample set s_B at $\mathbf{r}$. Sample sets s_A and s_B can be implemented as two separate textures or as one texture with two channels. We have not found an appreciable difference between these two methods. We present the complete Cg source of the proposed linear box spline resampling algorithm in the appendix.

We compare the rendering speed of our trilinearly evaluated linear box spline scheme to the latest method, that uses twice as many nearest neighbor fetches [13]. We also give a comparison to the major competitive methods: to the BCC B-spline [8] and to the BCC DC-spline [10]. Comprehensive analysis of the numerical accuracy, and visual quality of these splines are out of the scope of this paper. We refer the interested reader to [6, 10, 13] for a more thorough overview.

The number of texture lookups and the arithmetic costs differ for each filter (see Table 1). The arithmetic cost of the trilinear B-spline filtering is practically negligible [6, 8], the DC-spline filtering has moderate addressing overhead [10], while the trilinear and nearest neighbor linear box spline schemes have the highest number of floating point operations [13]. Concerning the number of texture fetches, the trilinear B-spline and the trilinearly evaluated linear box spline are in the best position: they need only two lookups, while the linear box spline filtering needs four fetches, and the DC-spline filtering needs six lookups.

Filter	lookups	complexity
Lin. box spline (nearest)	4	high
Lin. box spline (linear)	2	high
Trilinear B-spline	2	low
Linear DC-spline	6	medium

Table 1: Number of texture lookups and the arithmetic cost of different reconstruction filters. These properties determine the rendering performance.

The skeleton of the ray caster application was the same for each filtering technique, only the filter kernels and the storage scheme of the BCC samples were altered. For the nearest neighbor box spline evaluation, the BCC samples are stored in a one-channel texture by shifting the samples of the second lattice by half a grid spacing in every dimension [13]. For the trilinear box spline scheme, for the BCC B-splines, and for the BCC DC-splines, the BCC samples were stored as two separate set of CC samples as a two-channel texture.

5.1 Rendering Speed

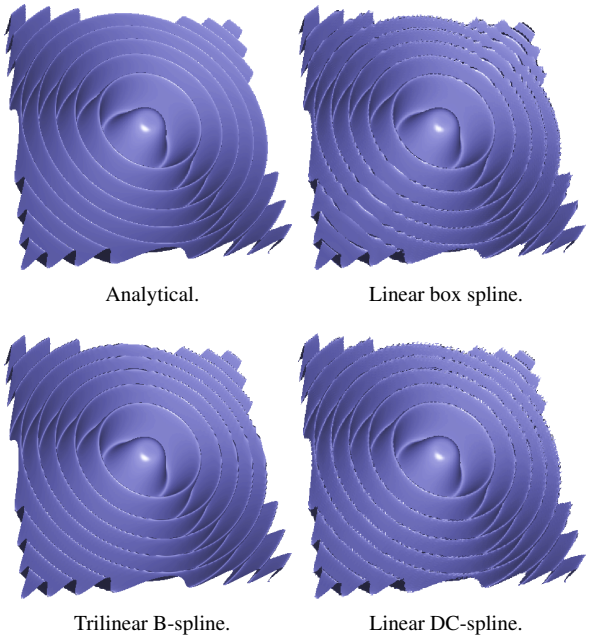
We rendered four data sets of different voxel counts at an image resolution of 512×512. The analytically defined Marschner-Lobb test signal was sampled at $64^3\times 2$ BCC resolution. The other three data sets are well-known CT scans reconstructed originally on a CC lattice. To get a BCC representation of them, we employed a frequency-domain upsampling [7].

The viewing rays were evaluated in front-to-back order, which enabled us to use early ray termination. The first-hit isosurfaces were shaded by the Blinn-Phong model using gradients calculated from central differences. The ray marching step and the central differencing step were adjusted to the voxel size of the data sets.

The renderings of the Marschner-Lobb test signal are illustrated in Figure 3. Note that the linear box spline introduces postaliasing artifacts along the diagonal directions, while the artifacts produced by the linear DC-spline or the trilinear B-spline are less apparent.

To get relevant rendering speeds, we chose the middle-aged NVIDIA Geforce 8700 GPU for our experiments. The observed frame rates are illustrated in Table 2. According to our prior expectations, the frame rates depended on the number of samples fetched, the algorithmic complexity of the filter kernel, the resolution of the volume, and the distance of the iso-surface from the image plane.

We can confirm the observation, that the frame rates get similar as the number of voxels increases with appropriately decreasing the sampling distance. Possibly, the texture fetches become the bottleneck of the rendering pipeline. This can be the reason why the DC-spline results in the lowest frame rates for the highest voxel counts.



Analytical. Linear box spline.

Trilinear B-spline. Linear DC-spline.

Figure 3: Renderings of the analytical Marschner-Lobb test signal and its sampled representations at $64^3 \times 2$ reconstructed by different resampling filters.

Data set	$M_{\Xi_{BCC}^{1,nearest}}$	$M_{\Xi_{BCC}^{1,linear}}$	β^1	χ_{BCC}^1
ML	21.03	22.90	53.64	21.75
Engine	16.28	17.18	41.82	16.42
Carp	9.22	10.00	25.07	9.19
Xmas Tree	5.77	6.19	6.57	4.61

Table 2: Frame rates in frames per second for different reconstruction filters and popular data sets: the Marschner-Lobb test signal sampled at $64^3 \times 2$, the "Engine Block" at $256^2 \times 110 \times 2$, the "Carp" at $256^2 \times 512 \times 2$, and the "Christmas Tree" at $512 \times 499 \times 512 \times 2$.

On the other hand, for low and moderate volume resolutions, the arithmetic complexity seems to be more important than the number of texture fetches. It is interesting to note that the concept of applying linear fetches instead of nearest neighbor ones [20] does not always pay off. We think that the texture cache operates very well for filters with a narrow support. This might explain that the nearest neighbor version and the linear version of the linear box spline filtering as well as the linear DC-spline filtering with even six samples attain similar frame rates, while the trilinear B-spline holds a towering lead in performance.

6 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a GPU evaluation scheme for the linear BCC box spline filtering exploiting the hardwired trilinear texture fetching. This result enabled us to make a fair comparison of the linear box spline, the BCC B-spline, and the BCC DC-spline in terms of their performance. We found that, in general, the proposed linear evaluation scheme operates slightly faster than the evaluation scheme with nearest neighbor fetches [13]. However, using an optimized GPU implementation, the trilinear B-spline can still achieve the best performance, as it takes the minimum number of samples with the lowest arithmetic cost. Since the texture fetches become more expensive when the support of the filter gets wider or the resolution of the volume increases, we plan to develop a similar scheme for the quintic box spline for the BCC lattice.

ACKNOWLEDGMENTS

This project was supported by Mediso Medical Imaging Systems, the Hungarian National Office for Research and Technology (Project ID: TECH 08/A2), and the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002). This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project.

REFERENCES

[1] T. Blu, P. Thévenaz, and M. Unser. Generalized interpolation: Higher quality at no additional cost. In *Proceedings of IEEE International Conference on Image Processing*, pages 667–671, 1999.

[2] Dutta Roy S. C. and Kumar B. *Handbook of Statistics*, volume 10, chapter Digital Differentiators, pages 159–205. Elsevier Science Publishers B. V., N. Holland, 1993.

[3] I. Carlbom. Optimal filter design for volume reconstruction and visualization. In *Proceedings of the 4th conference on Visualization*, pages 54–61. IEEE Computer Society, 1993.

[4] B. Csébfalvi. Prefiltered gaussian reconstruction for high-quality rendering of volumetric data sampled on a body-centered cubic grid. In *Proceedings of IEEE Visualization*, pages 311–318, 2005.

[5] B. Csébfalvi. BCC-splines: Generalization of B-splines for the body-centered cubic lattice. *Journal of WSCG 16, 1–3 (2008)*, pages 81–88, 2008.

[6] B. Csébfalvi. An evaluation of prefiltered B-spline reconstruction for quasi-interpolation on the body-centered cubic lattice. *IEEE Transactions on Visualization and Computer Graphics 16, 3 (2010)*, pages 499–512, 2010.

[7] B. Csébfalvi and B. Domonkos. Frequency-domain upsampling on a body-centered cubic lattice for efficient and high-quality volume rendering. In *Proceedings of Vision, Modeling, and Visualization*, pages 225–232, 2009.

[8] B. Csébfalvi and M. Hadwiger. Prefiltered B-spline reconstruction for hardware-accelerated rendering of optimally sampled volumetric data. In *Proceedings of VMV*, pages 325–332, 2006.

[9] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*, volume 98. Springer-Verlag, 1993.

[10] B. Domonkos and B. Csébfalvi. DC-splines: Revisiting the trilinear interpolation on the body-centered cubic lattice. In *Proceedings of VMV*, pages 275–282, 2010.

[11] A. Entezari. *Optimal Sampling Lattices and Trivariate Box Splines*. PhD thesis, Simon Fraser University, Vancouver, Canada, July 2007.

[12] A. Entezari, D. Van De Ville, and T. Möller. Practical box splines for reconstruction on the body centered cubic lattice. *IEEE Trans. on Vis. and Computer Graphics*, 14(2):313–328, 2008.

[13] B. Finkbeiner, A. Entezari, D. Van De Ville, and T. Möller. Efficient volume rendering on the body centered cubic lattice using box splines. *Computers and Graphics*, 34(4):409–423, 2010.

[14] L. Kobbelt. Stable evaluation of box splines. *Numerical Algorithms*, 14, 1996.

[15] O. Mattausch. Practical reconstruction schemes and hardware-accelerated direct volume rendering on bodycentered cubic grids. Master's thesis, Vienna University of Technology, 2003.

[16] M. Mirzargar and A. Entezari. Voronoi splines. *IEEE Transactions on Signal Processing*, 58:4572–4582, Sept 2010.

[17] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *Proceedings of the 8th conference on Visualization*, pages 19–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[18] T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proceedings of the IEEE symposium on Volume visualization*, pages 143–151, New York, NY, USA, 1998. ACM.

[19] D. P. Petersen and D. Middleton. Sampling and reconstruction of wave-number-limited functions in n-dimensional euclidean spaces. *Information and Control*, 5(4):279–323, 1962.

[20] C. Sigg and M. Hadwiger. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 313–329. Addison- Wesley, 2005.

[21] T. Theußl, T. Möller, and M. E. Gröller. Optimal regular volume sampling. In *Proceedings of the conference on Visualization*, pages 91–98. IEEE Computer Society, 2001.

[22] T. Theußl, T. Möller, and M. E. Gröller. Reconstruction schemes for high quality raycasting of the body-centered cubic grid. Technical Report TR-186-2-02-11, Institute of Computer Graphics and Algorithms, TU Vienna, Dec 2002.

[23] D. Van De Ville, T. Blu, M. Unser, W. Philips, I. Lemahieu, and R. Van de Walle. Hex-Splines: A novel spline family for hexagonal lattices. *IEEE Transactions on Image Processing*, 13(6):758–772, 2004.

CG SHADER CODE

```
uniform float3    Size;
uniform sampler3D Volume;

// Handle removable singularity
#define DIV( A, B ) \
  ( abs(B) ? ( A ) / ( B ) : 0.0 )

// I^th coordinate of unit vector e_π(1)
#define E_PI_1( I ) ( a.I == x )

// I^th coordinate of unit vector e_π(3)
#define E_PI_3( I ) \
  ( a.I == z && a.I != x && a.I != y )

// Unit vector e_π(J)
#define E_PI( J ) float3( E_PI_ ## J( x ), \
  E_PI_ ## J( y ), E_PI_ ## J( z ) )

// Fetching a trilinear sample from Λ_CC_A at R
#define S_A( R ) \
  tex3D(Volume, (r_base + (R) + 0.5) / Size).r

// Fetching a trilinear sample from Λ_CC_B at R
#define S_B( R ) \
  tex3D( Volume, ( r_base + (R) ) / Size ).a

float linearBoxSpline( float3 texCoords ) {
  // Resampling point r
  float3 r = texCoords * Size - 0.5;

  // Nearest lattice point of Λ_CC_A
  float3 r_base = round( r );

  // Relative coordinates d,
  // their absolute values a, and signs s
  float3 d = r - r_base;
  float3 a = abs( d );
  float3 s = sign( d );

  // Sorting a by its components
  float x = max( a.x, max( a.y, a.z ) );
  float z = min( a.x, min( a.y, a.z ) );
  float y = a.x + a.y + a.z - x - z;

  // Fetching from sample sets Λ_CC_A and Λ_CC_B
  float two_y = 2.0 * y;
  float tA = DIV( x - y, 1.0 - two_y );
  float tB = DIV( z - y, two_y );
  float fA = S_A( tA * s * E_PI(1) );
  float fB = S_B(s * ( 0.5 + tB * E_PI(3) ));

  // Linear interpolation of the two samples
  return lerp( fA, fB, two_y );
}
```

High-Performance Terrain Rendering Using Hardware Tessellation

Egor Yusov
Intel Corporation
30 Turgeneva street,
603024, Russia, Nizhny Novgorod
egor.a.yusov@intel.com

Maxim Shevtsov
Intel Corporation
30 Turgeneva street,
603024, Russia, Nizhny Novgorod
maxim.y.shevtsov@intel.com

ABSTRACT

In this paper, we present a new terrain rendering approach, with adaptive triangulation performed entirely on the GPU via tessellation unit available on the DX11-class graphics hardware. The proposed approach avoids encoding of the triangulation topology thus reducing the CPU burden significantly. It also minimizes the data transfer overhead between host and GPU memory, which also improves rendering performance. During the preprocessing, we construct a multiresolution terrain height map representation that is encoded by the robust compression technique enabling direct error control. The technique is efficiently accelerated by the GPU and allows the trade-off between speed and compression performance. At run time, an adaptive triangulation is constructed in two stages: a coarse and a fine-grain one. At the first stage, rendering algorithm selects the coarsest level patches that satisfy the given error threshold. At the second stage, each patch is subdivided into smaller blocks which are then tessellated on the GPU in the way that guarantees seamless triangulation.

Keywords

Terrain rendering, DX11, GPU, adaptive tessellation, compression, level of detail.

1. INTRODUCTION

Despite the rapid advances in the graphics hardware, high geometric fidelity and real-time large scale terrain visualization is still an active research area. The primary reason is that the size and resolution of digital terrain models grow at a significantly higher rate than the graphics hardware can manage. Even the modest height map can easily exceed the available memory of today's highest-end graphics platforms. So it is still important to dynamically control the triangulation complexity and reduce the height map size to fit the hardware limitations and meet real-time constraints.

To effectively render large terrains, a number of dynamic multiresolution approaches as well as data compression techniques have been developed in the last years. These algorithms typically adapt the terrain tessellation using local surface roughness criteria together with the view parameters. This allows dramatic reduction of the model complexity without significant loss of visual accuracy. Brief overview of different terrain rendering approaches is given in the following section. In the previous methods, the adaptive triangulation was usually constructed by the CPU and then transferred to the GPU for rendering. New capabilities of DX11-class graphics hardware enable new approach, when adaptive terrain tessellation is built entirely on the GPU. This reduces the memory storage requirements together with the CPU load. It also reduces the amount of data to be transferred from the main memory to the GPU that again results in a higher rendering performance.

2. RELATED WORK

Many research papers about adaptive view-dependent triangulation construction methods were published in the last years. Refer to a nice survey by R. Pajarola and E. Gobbetti [PG07].

Early approaches construct triangulated irregular networks (TINs). Exploiting progressive meshes for terrain simplification [Hop98] is one specific example. Though TIN-based methods do minimize the amount of triangles to be rendered for a given error bound, they are too computationally and storage

demanding. More regular triangulations such as bintree hierarchies [LKR+96, DWS+97] or restricted quad trees [Paj98] are faster and easier to implement for the price of slightly more redundant triangulation.

Recent approaches are based on techniques that fully exploit the power of modern graphics hardware. CABTT algorithm [Lev02] by J. Levenberg as well as BDAM [CGG+03a] and P-BDAM [CGG+03b] methods by Cignoni et al exploit bintree hierarchies of pre-computed triangulations or batches instead of individual triangles. Geometry clipmaps approach [LH04] renders the terrain as a set of nested regular grids centered about the viewer, allowing efficient GPU utilization. The method exploits regular grid pyramid data structure in conjunction with the lossy image compression technique [Mal00] to dramatically reduce the storage requirements. However, the algorithm completely ignores local surface features of the terrain and provides no guarantees for the error bound, which becomes especially apparent on high-variation terrains.

Next, C-BDAM method, an extension of BDAM and P-BDAM algorithms, was presented by Gobbetti et al in [GMC+06]. The method exploits a wavelet-based two stage near-lossless compression technique to efficiently encode the height map data. In C-BDAM, uniform batch triangulations are used which do not adapt to local surface features. Regular triangulations typically generate significantly more triangles and unreasonably increase the GPU load.

Terrain rendering method presented by Schneider and Westermann [SW06] partitions the terrain into square tiles and builds for each tile a discrete set of LODs using a nested mesh hierarchy. Following this approach, Dick et al proposed the method for tile triangulations encoding that enables efficient GPU-based decoding [DSW09].

All these methods either completely ignore local terrain surface features (like [LC03, LH04, GMC+06]) for the sake of efficient GPU utilization, or pre-compute the triangulations off-line and then just load them during rendering [CGG+03a, CGG+03b]. For the case of compressed data, GPU can also be used for geometry decompressing as well [SW06, DSW09].

By the best of our knowledge, none of the previous methods take an advantage of the tessellation unit exposed by the latest DX11-class graphics hardware for precise yet adaptive (view-dependent) terrain tessellation.

3. CONTRIBUTION

The main contribution is a novel terrain rendering approach, which combines efficiency of the chunk-based terrain rendering with the accuracy of fine-grain triangulation construction methods. In contrast to the previous approaches, our adaptive view-dependent triangulation is constructed entirely on the GPU using hardware-supported tessellation. This offloads computations from the CPU while also reduces expensive CPU-GPU data transfers. We also propose fast and simple GPU-accelerated compression technique for progressively encoding multiresolution hierarchy that enables direct control of a reconstruction precision.

Algorithm Overview

To achieve real-time rendering and meet the hardware limitations, we exploit the LOD technique. To create various levels of detail, during the preprocessing, a multiresolution hierarchy is constructed by recursively downsampling the initial data and subdividing it into overlapping patches. In order to reduce the memory requirements, the resulting hierarchy is then encoded using simple and efficient compression algorithm described in section 4.

Constructing adaptive terrain model to be rendered is a two-stage process. The first stage is the coarse per-patch LOD selection: the rendering algorithm selects the coarsest level patches that tolerate the given screen-space error. They are cached in a GPU memory and due to the frame-to-frame coherence are re-used for a number of successive frames. On the second stage, a fine-grain LOD selection is performed: each patch is seamlessly triangulated using hardware. For this purpose, each patch is subdivided into the equal-sized smaller blocks that are independently triangulated by the GPU-supported tessellation unit, as described in section 5.

Experimental results are given in section 6. Section 7 concludes the paper.

4. BUILDING COMPRESSED MULTIRESOLUTION TERRAIN REPRESENTATION

Patch Quad Tree

The core structure of the proposed multiresolution model is a quad tree of square blocks (hereinafter referred to as *patches*). This structure is commonly used in real-time terrain rendering systems [Ulr00, DSW09].

The patch quad tree is constructed at the preprocess stage. At the first step, a series of coarser height maps is built. Each height map is the downsampled version of the previous one (fig. 1). At the next step, the patch quad tree itself is constructed by subdividing each level into $(2^n+1)\times(2^n+1)$ square blocks (65x65, 129x129, 257x257 etc.), refer to fig. 2.

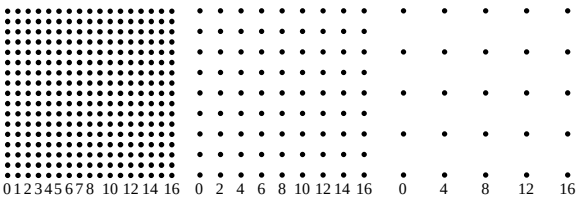


Figure 1. Downsampling initial height map.

Each patch in the quad tree hierarchy approximates the same area as its four children but with lower accuracy. To eliminate cracks, each patch shares one-sample boundary with its neighbors (hence 2^n+1 size).

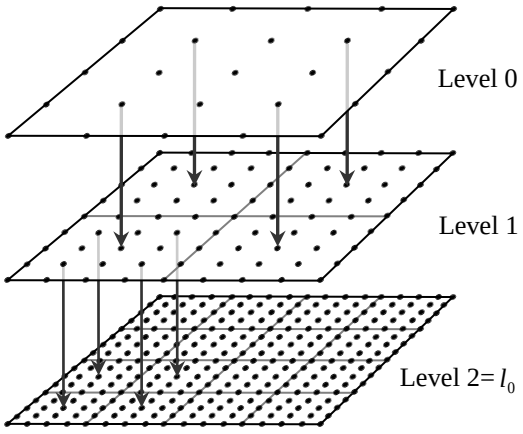


Figure 2. Patch quad tree.

The hierarchy is progressively encoded in a top-down order such that each patch's reconstruction error in L^∞ metric is bounded by the given error tolerance.

Quantizing Height Maps

Let's denote a sample in the *l*-th level located at the (*i*, *j*) position by $h_{i,j}^{(l)}$. Note that since level *l*-1 is simply the downsampled version of the level *l*, the following relation is always true: $h_{i,j}^{(l-1)} \equiv h_{2i,2j}^{(l)}$.

During the compression process, each level l of the hierarchy is quantized using a uniform quantizer with a dead zone (see fig. 3) as follows:

$$\hat{h}_{i,j}^{(l)} = \left\lfloor (h_{i,j}^{(l)} + \delta_l)/(2\delta_l) \right\rfloor \cdot 2\delta_l$$

where $\lfloor x \rfloor$ is rounding to the largest integer that is less than or equal to *x*, $\hat{h}_{i,j}^{(l)}$ is the quantized value, $\delta_l = \delta_{l_0} 2^{(l_0 - l)}$ is the maximum reconstruction error for the level *l*; l_0 is the finest resolution level number and $\delta_{l_0} > 0$ is the user-defined error tolerance for the finest level. Since our compression scheme is lossy, we assume that $\delta_{l_0} > 0$.

Quantized (integer) values $q_{i,j}^{(l)} = Q_l(h_{i,j}^{(l)})$ where $Q_l(h) = \left\lfloor (h+\delta_l)/(2\delta_l) \right\rfloor$ are lossless encoded as described below. The decoder reconstructs values as:

$$\hat{h}_{i,j}^{(l)} = 2\delta_l q_{i,j}^{(l)}$$

This quantization rule assures that for the *l*-th level, the maximum error is bounded by the δ_l:

$$\max_{i,j} | \hat{h}_{i,j}^{(l)} - h_{i,j}^{(l)} | \le \delta_l$$

The quantized values $\{q_{i,j}^{(0)}\}$ of the coarsest patch (located at the level 0) are encoded using adaptive arithmetic coding [WNC87]. The remaining patches are then progressively encoded as described in the following subsection.

Progressively Encoding Quantized Height Maps

Let us consider a patch's quantized height map $\hat{H}_P^{(l-1)}$ at the level *l*-1, and its 4 children joined height map $\hat{H}_C^{(l)}$ at the level *l*. Note that the first height map is $(2^n+1)\times(2^n+1)$ in size, while the second one is $(2\cdot 2^n+1)\times(2\cdot 2^n+1)$, both covering the same area. As it can be seen from fig. 1 and 2 (see also fig. 4), $\hat{H}_C^{(l)}$ shares the samples located at the even positions ((0,0), (0,2), (2,0) and so on) with $\hat{H}_P^{(l-1)}$. That is, the reconstructed sample $\hat{h}_{i,j}^{(l-1)}$ from the parent patch's height map $\hat{H}_P^{(l-1)}$ corresponds to the sample $\hat{h}_{2i,2j}^{(l)}$ in the $\hat{H}_C^{(l)}$. However $\hat{h}_{i,j}^{(l-1)}$ approximates the original (exact) value with the 2x lower accuracy than $\hat{h}_{2i,2j}^{(l)}$ should approximate and thus needs to be refined:

$$| \hat{h}_{i,j}^{(l-1)} - h_{i,j}^{(l-1)} | \le \delta_{l-1} = 2\delta_l$$

$$| \hat{h}_{2i,2j}^{(l)} - h_{2i,2j}^{(l)} | \le \delta_l$$

Recall that $h_{i,j}^{(l-1)} \equiv h_{2i,2j}^{(l)}$.

Our compression scheme consists of two steps. At the first step, we refine common samples of $\hat{H}_C^{(l)}$ and $\hat{H}_P^{(l-1)}$ (filled circles in fig. 4) to the required accuracy δ_l. At the second step, we encode the remaining samples (dotted circles in fig. 4) by interpolating the refined samples and encoding the prediction errors. Let's denote *R* to be the set of refined samples positions from $\hat{H}_C^{(l)}$ and *I* to be the set of interpolated samples positions:

$$R = \{(i,j) : \hat{h}_{i,j}^{(l)} \in \hat{H}_C^{(l)} \wedge i = 2s,\ j = 2t,\ s,t \in Z\}$$

$$I = \{(i,j) : \hat{h}_{i,j}^{(l)} \in \hat{H}_C^{(l)} \wedge (i,j) \notin R\}$$

To refine samples from *R*, we exploit the following observation: the refined sample $\hat{h}_{2i,2j}^{(l)}$ (from $\hat{H}_C^{(l)}$) corresponding to the sample $\hat{h}_{i,j}^{(l-1)}$ (from $\hat{H}_P^{(l-1)}$) can only take one of the following 3 values (see fig. 3):

$$\hat{h}_{2i,2j}^{(l)} \in \{\hat{h}_{i,j}^{(l-1)} - 2\delta_l,\ \hat{h}_{i,j}^{(l-1)},\ \hat{h}_{i,j}^{(l-1)} + 2\delta_l\}.$$

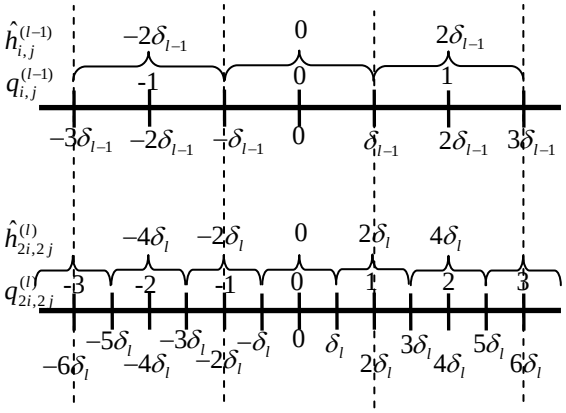


Figure 3. Quantizing two successive levels.

This also means that if $\hat{h}_{i,j}^{(l-1)}$ is encoded by the quantized value $q_{i,j}^{(l-1)}$, then corresponding $q_{2i,2j}^{(l)}$ can only take one of the following 3 values:

$$q_{2i,2j}^{(l)} \in \{2q_{i,j}^{(l-1)} - 1,\ 2q_{i,j}^{(l-1)},\ 2q_{i,j}^{(l-1)} + 1\}$$

Since $q_{i,j}^{(l-1)}$ is known, encoding the $q_{2i,2j}^{(l)}$ requires only 3 symbols: -1, 0 or $+1$. These symbols are encoded using adaptive arithmetic coding [WNC87].

At the second step, we encode the remaining samples located at positions from *I* in $\hat{H}_C^{(l)}$ (dotted circles in fig. 4). This is done by predicting the sample's value from the refined samples and by encoding the prediction error.

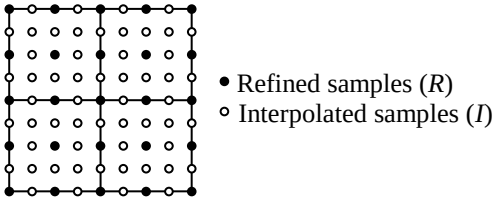


Figure 4. Refined and interpolated samples of the child patches joined height map $\hat{H}_C^{(l)}$.

For the sake of GPU-acceleration, we exploit bilinear predictor $P_R(\hat{h}_{i,j}^{(l)})$ that calculates predicted value of $\hat{h}_{i,j}^{(l)}$ as a weighted sum of 4 refined samples located at the neighboring positions in *R*. We then calculate the prediction error as follows:

$$d_{i,j}^{(l)} = Q_l(P_R(\hat{h}_{i,j}^{(l)})) - q_{i,j}^{(l)},\ (i,j) \in I$$

Magnitudes and signs of the resulting prediction errors $d_{i,j}^{(l)}$ are then separately encoded using adaptive arithmetic coding.

As it was already discussed, symbols being used during described compression process are encoded with the technique described in [WNC87]. We exploit adaptive approach that learns the statistical properties of the input symbol stream on the fly. This is implemented as a histogram which counts corresponding symbol frequencies (see [WNC87] for details). Note that simple context modeling can improve the compression performance with minimal algorithmic complexity increase.

During the preprocessing, the whole hierarchy is recursively traversed starting from the root (level 0) and the proposed encoding process is repeated for each patch.

The proposed compression scheme enables direct control of the reconstruction precision in L^∞ error metric: it assures that the maximum reconstruction error of a terrain block at level *l* of the hierarchy is no more than δ_l. For comparison, compression method [Mal00] used in geometry clipmaps [LH04] does not provide a guaranteed error bound in L^∞ metric. C-BDAM [GMC+06] exploits sophisticated two-stage compression scheme to assure the given error tolerance. This provides higher compression ratios but is more computationally expensive than the presented scheme. Moreover, as we will show in the next section, our technique can be efficiently accelerated using the GPU.

Calculating Guaranteed Patch Error Bound

During the quad tree construction, each patch in the hierarchy is assigned a world space approximation error. It conservatively estimates the maximum geometric deviation of the patch's reconstructed height map from the underlying original full-detail height map. This value is required at the run time to estimate the screen-space error and to construct the patch-based adaptive model, which approximates the terrain with the specified screen-space error.

Let's denote the patch located at the level *l* of the quad tree at the (*m*, *n*) position by the $P_{m,n}^{(l)}$ and its upper error bound by the $Err(P_{m,n}^{(l)})$. To calculate $Err(P_{m,n}^{(l)})$, we first calculate *approximation error* $Err_{Appr}(P_{m,n}^{(l)})$, which is the upper bound of the maximum distance from the patch's precise height map to the samples of the underlying full-detail (level

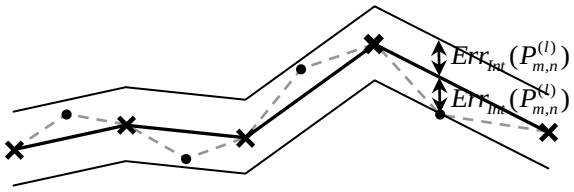
l_0) height map. It is recursively calculated using the same method as used in ROAM [DWS+97] to calculate the nested wedgie thickness:

$$Err_{Appr}(P_{m,n}^{(l_0)}) = 0$$

$$Err_{Appr}(P_{m,n}^{(l)}) = Err_{Int}(P_{m,n}^{(l)}) + \max_{s,t=\pm 1}\{Err_{Appr}(P_{2m+s,2n+t}^{(l+1)})\},$$

$$l = l_0 - 1,..0$$

where $Err_{Int}(P_{m,n}^{(l)})$ is the maximum geometric deviation of the linearly interpolated patch's height map from its children height maps. Two-dimensional illustration for determining $Err_{Int}(P_{m,n}^{(l)})$ is given in fig. 5.



- Child patches' (level *l*) height map samples
- ✖ Parent patch's (level *l*-1) height map samples

Figure 5. Patch's height map interpolation error.

Since for the patch $P_{m,n}^{(l)}$, the reconstructed height map deviates from the exact height map by at most δ_l, the final patch's upper error bound is given by:

$$Err(P_{m,n}^{(l)}) = Err_{Appr}(P_{m,n}^{(l)}) + \delta_l$$

5. CONSTRUCTING VIEW-DEPENDENT ADAPTIVE MODEL

The proposed level-of-detail selection process consists of two stages. The first stage is the coarse LOD selection which is done on a per-patch level: an unbalanced patch quad tree is constructed with the leaf patches satisfying the given error tolerance. On the second stage, the fine-grain LOD selection is performed, at which each patch is precisely triangulated using the hardware tessellation unit.

Coarse Level of Detail Selection

The coarse LOD selection is performed similar to other quad tree-based terrain rendering methods. For this purpose, an unbalanced patch quad tree is maintained. It defines the block-based adaptive model, which approximates the terrain with the specified screen-space error.

The unbalanced quad tree is cached in a GPU memory and is updated according to the results of comparing patch's screen-space error estimation $Err_{Scr}(P_{m,n}^{(l)})$ with the user-defined error threshold ε. Since we already have the maximum geometric error for the vertices within a patch, $Err_{Scr}(P_{m,n}^{(l)})$ can be calculated using standard LOD formula for conservatively determining the maximum screen-space vertex error (see [Ulr00, Lev02]):

$$Err_{Scr}(P_{m,n}^{(l)}) = \gamma \frac{Err(P_{m,n}^{(l)})}{\rho(c, V_{m,n}^{(l)})}$$

where $\gamma = \frac{1}{2}\max(R_h \cdot ctg(\varphi_h/2),\ R_v \cdot ctg(\varphi_v/2))$, R_h and R_v are horizontal and vertical resolutions of the view port, φ_h and φ_v are the horizontal and vertical camera fields of view, and $\rho(c, V_{m,n}^{(l)})$ is the distance from the camera position *c* to the closest point on the patch's bounding box $V_{m,n}^{(l)}$.

Tessellation Blocks

During the fine-grain LOD selection, each patch in the unbalanced patch quad tree is adaptively triangulated using the GPU. For this purpose, each patch is subdivided into the small equal-sized blocks that we call *tessellation blocks*. For instance, a 65×65 patch can be subdivided into the 4×4 grid of 17×17 tessellation blocks or into the 8×8 grid of 9×9 blocks etc. Detail level for each tessellation block is determined independently by the hull shader: the block can be rendered in the full resolution (fig. 6, left) or in the resolution reduced by a factor of 2^d, $d = 1,2,\ldots$ (fig. 6, center, right).

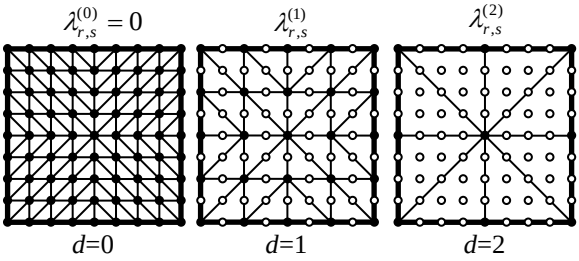


Figure 6. Triangulations of a 9×9 tessellation block.

To determine the degree of simplification for each block, we calculate a series of block errors. These errors represent the deviation of the block's simplified triangulation from the patch's height map samples, covered by the block but not included into the simplified triangulation (dotted circles in fig. 6).

Let's denote the error of the tessellation block located at the (*r*, *s*) position in the patch, whose triangulation is simplified by a factor of 2^d by $\lambda_{r,s}^{(d)}$. The tessellation block errors $\lambda_{r,s}^{(d)}$ are computed as follows:

$$\lambda_{r,s}^{(d)} = \max_{v \notin T_{r,s}^{(d)}} \rho(v, T_{r,s}^{(d)}),\ d = 1,2,\ldots$$

where $T_{r,s}^{(d)}$ is the tessellation block (*r*,*s*) triangulation simplified by a factor of 2^d and $\rho(v, T_{r,s}^{(d)})$ is the vertical distance from the vertex *v* to the triangulation $T_{r,s}^{(d)}$. Two and four times simplified triangulations as well as these samples (dotted circles) of the patch's height map that are used to calculate $\lambda_{r,s}^{(1)}$ and $\lambda_{r,s}^{(2)}$ are shown in fig. 6 (center and right images correspondingly).

To get the final error bound for the tessellation block, it is necessary to take into account the patch's error bound. This final error bound hereinafter is referred to as $\Lambda_{r,s}^{(d)}$ and is calculated as follows:

$$\Lambda_{r,s}^{(d)} = \lambda_{r,s}^{(d)} + Err(P_{m,n}^{(l)})$$

In our particular implementation, we calculate errors for 4 simplification levels such that tessellation block triangulation can be simplified by a maximum factor of $(2^4)^2 = 256$. This enables us to store the tessellation block errors as a 4-component vector.

Fine-Grain Level of Detail Selection

When the patch is to be rendered, it's necessary to estimate how much its tessellation blocks' triangulations can be simplified without introducing unacceptable error. This is done using the current frame's world-view-projection matrix. Each tessellation block is processed independently and for each block's edge, a tessellation factor is determined. To eliminate cracks, tessellation factors for shared edges of neighboring blocks must be computed in the same way. The tessellation factors are then passed to the tessellation stage of the graphics pipeline, which generates final triangulation.

Tessellation factors for all edges are determined identically. Let's consider some edge and denote its center by e_c. Let's define edge errors $\Lambda_{e_c}^{(d)}$ as the maximum error of the tessellation blocks sharing this edge. For example, block (*r*, *s*) left edge's errors are calculated as follows:

$$\Lambda_{e_c}^{(d)} = \max(\Lambda_{r-1,s}^{(d)}, \Lambda_{r,s}^{(d)}),\ d = 1,2,\ldots$$

Next let's define a series of segments in a world space specified by theirs end points $e_c^{(d),-}$ and $e_c^{(d),+}$ determined as follows:

$$e_c^{(d),-} = e_c - \Lambda_{e_c}^{(d)} / 2 \cdot e_z$$

$$e_c^{(d),+} = e_c + \Lambda_{e_c}^{(d)} / 2 \cdot e_z$$

where e_z is the world space *z* (up) axis unit vector.

Thus $e_c^{(d),-}$ and $e_c^{(d),+}$ define a segment of length $\Lambda_{e_c}^{(d)}$ directed along the *z* axis such that the edge centre e_c is located in the segment's middle.

If we project this segment onto the viewing plane using the world-view-projection matrix, we will get the edge screen space error estimation (fig. 7) given that the neighboring tessellation blocks are simplified by a factor of 2^d. We can then select the maximum simplification level *d* for the edge that does not lead to unacceptable error as follows:

$$d = \arg\max_d proj(\overline{e_c^{(d),-}, e_c^{(d),+}}) < \varepsilon$$

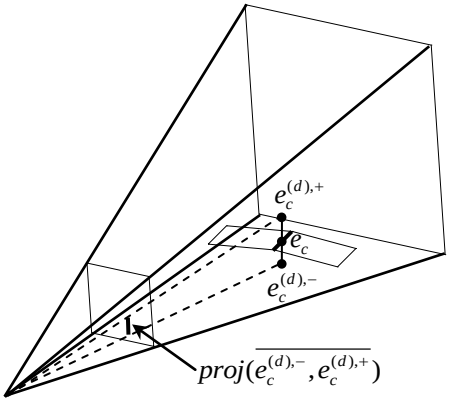


Figure 7. Calculating edge screen space error.

The same selection process is done for each edge. Tessellation factor for the block interior is then defined as the minimum of its edge tessellation factors. This method assures that tessellation factors for shared edges of neighboring blocks are computed equally and guarantees seamless patch triangulation. An example of a patch triangulation is given in fig. 8.

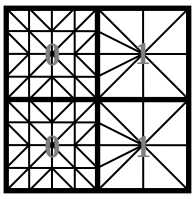


Figure 8. Seamlessly triangulated patch's tessellation blocks.

To hide gaps between neighboring patches, we exploit "vertical skirts" around the perimeter of each patch as proposed by T. Ulrich [Ulr00]. The top of the skirt matches the patch's edge and the skirt height is selected such that it hides all possible cracks.

Note that in contrast to all previous terrain simplification methods, all operations required to triangulate the patch are performed entirely on the GPU and does not involve any CPU computations.

Implementation Details

The presented algorithm was implemented with the C++ in an MS Visual Studio .NET environment.

In our system, the CPU decodes the bit stream in parallel to the rendering thread and all other tasks are done on the GPU. To facilitate GPU-accelerated decompression, we support several temporary textures. The first one is $(2^n+1)\times(2^n+1)$ 8-bit texture T_R that is populated with the parent patch's refinement labels (-1, 0 or $+1$) from *R*. The second one is $(2\cdot 2^n+1)\times(2\cdot 2^n+1)$ 8-bit texture T_I storing prediction errors $d_{i,j}^{(l)}$ for samples from *I*. GPU-part of the decompression is done in two steps:

- At the first step, parent patch height map is refined by rendering to the temporary texture T_P.
- At the second step, child patch height maps are rendered.

During the second step, T_P is filtered using hardware-supported bilinear filtering, interpolation errors are loaded from T_I and added to the interpolated samples from T_P.

Patch's height and normal maps as well as the tessellation block errors are stored as texture arrays. A list of unused subresources is supported. When patch is created, we find unused subresource in the list and release it when the patch is destroyed. Tessellation block errors as well as normal maps are computed on the GPU when the patch is created by rendering to the appropriate texture array element.

Exploiting texture arrays enables the whole terrain rendering using single draw call with instancing. The per-instance data contains patch location, level in the hierarchy and the texture index. Patch rendering hull shader calculates tessellation factor for each edge and passes the data to the tessellator. Tessellator generates topology and domain coordinates that are passed to the domain shader. Domain shader calculates world space position for each vertex and fetches the height map value from the appropriate texture array element. The resulting triangles then pass in a conventional way via rasterizer.

6. EXPERIMENTAL RESULTS AND DISCUSSION

To test our system, we used 16385×16385 height map of the Puget Sound sampled at 10 meter spacing, which is used as the common benchmark and is available at [PS]. The raw data size (16 bps) is 512 MB. The compression and run-time experiments were done on a workstation with the following hardware configuration: single Intel Core i7 @2.67; 6.0 GB RAM; NVidia GTX480.

The data set was compressed to 46.8 MB (11:1) with 1 meter error tolerance. For comparison, C-BDAM method, which exploits much more sophisticated approach, compressed the same data set to 19.2 MB (26:1) [GMC+06]. Note that in contrast to C-BDAM, our method enables hardware-based decompression. Note also that in practice we compress extended $(2^n+3)\times(2^n+3)$ height map for each patch for the sake of seamless normal map generation. As opposed to compressing conventional diffuse textures, height maps usually require less space. That is why we believe that provided 11x compression rate is a good justification for the quality of our algorithm.

During our run-time experiments, the Puget Sound data set was rendered with 2 pixels screen space error tolerance at 1920x1200 resolution (fig. 10). We compared the rendering performance of our method with our implementation of the chunked LOD approach [Ulr00]. As fig. 10 shows, the data set was rendered at **607** fps on average with minimum at **465** fps with the proposed method. When the same terrain was rendered with our method but without exploiting instancing and texture arrays described previously, the frame rate was almost **2x** lower. As fig. 10 shows, our method is more than **3.5x** faster than the chunked LOD approach.

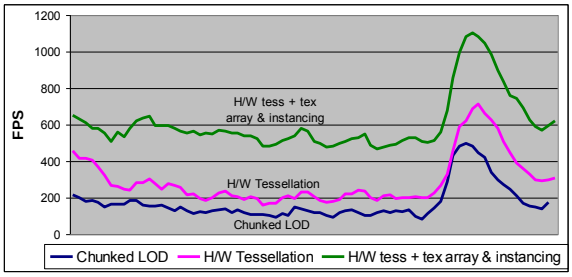


Figure 10. Rendering performance at 1920×1200 resolution.

Our experiments showed that the optimal tessellation block size that provides the best performance is 8×8. Other interesting statistics for this rendering experiment is that approximately 1024 of 128×128 patches were kept in GPU memory (only ~200 of them were rendered per frame on average). Each height map was stored with 16 bit precision. All patches demanded just 32 MB of the GPU memory. We also exploited normal map compressed using BC5, which required additional 16 MB of data. Diffuse maps are not taken into account because special algorithms that are behind the scope of this work are designed to compress them. However, the same quad tree-based subdivision scheme can be integrated with our method to handle diffuse texture.

Since our method enables using small screen space error threshold (2 pixels or less), we did not observe any popping artifacts during our experiments even

though there is no morph between successive LODs in our current implementation.

In all our experiments, the whole compressed hierarchy easily fitted into the main memory. However, our approach can be easily extended for the out-of-core rendering of arbitrary large terrains. In this case, the whole compressed multiresolution representation would be kept in a repository on the disk or a network server, as for example in the geometry clipmaps. This would allow on-demand extraction from the repository rather accessing the data directly in the memory.

7. CONCLUSION AND FUTURE WORK

We presented a new real-time large-scale terrain rendering technique, which is based on the exploitation of the hardware-supported tessellation available in modern GPUs. Since triangulation is performed entirely on the GPU, there is no need to encode the triangulation topology. Moreover, the triangulation is precisely adapted to each camera position. To reduce the data storage requirements, we use robust compression technique that enables direct control over the reconstruction precision and is also accelerated by the GPU.

We consider support for dynamic terrain modifications as a future work topic. Since the triangulation topology is constructed entirely on the GPU, it would require only updating the tessellation block errors, and the triangulation will be updated accordingly. Another possible direction is to extend the presented algorithm for rendering arbitrary high-detailed 2D-parameterized surfaces.

8. REFERENCES

[CGG+03a] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum, Vol. 22, No. 3, pp. 505–514, 2003.

[CGG+03b] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. Planet-sized batched dynamic adaptive meshes (P-BDAM). In Proc. IEEE Visualization, pp. 147–154, 2003.

[DSW09] Dick, C., Schneider, J., and Westermann, R. Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering. In Computer Graphics Forum, Vol. 28, No 1, pp. 67–83, 2009.

[DWS+97] Duchaineau, M., Wolinsky, M., Sigeti, D.E., Miller, M.C., Aldrich, C., and Mineev-Weinstein, M.B. ROAMing terrain: Real-time optimally adapting meshes. In Proc. IEEE Visualization, pp. 81–88, 1997.

[GMC+06] Gobbetti, E., Marton, F., Cignoni, P., Di Benedetto, M., and Ganovelli, F. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. Computer Graphics Forum, Vol. 25, No. 3, pp. 333–342, 2006.

[Hop98] Hoppe, H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In Proc. IEEE Visualization, pp. 35–42, 1998.

[LC03] Larsen, B.D., and Christensen, N.J. Real-time Terrain Rendering using Smooth Hardware Optimized Level of Detail. Journal of WSCG, Vol. 11, No. 2, pp. 282–289, 2003.

[Lev02] Levenberg, J. Fast view-dependent level-of-detail rendering using cached geometry. In Proc. IEEE Visualization, pp. 259–265, 2002.

[LKR+96] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L.F., Faust, N., and Turner, G.A. Real-time, continuous level of detail rendering of height fields. In Proc. ACM SIGGRAPH, pp. 109–118, 1996.

[LH04] Losasso, F., and Hoppe, H. Geometry clipmaps: Terrain rendering using nested regular grids. In Proc. ACM SIGGRAPH, pp. 769–776, 2004.

[Mal00] Malvar, H. Fast Progressive Image Coding without Wavelets. In Proceedings of Data Compression Conference (DCC ’00), Snowbird, UT, USA, pp. 243–252, 28-30 March 2000.

[Paj98] Pajarola, R. Large scale terrain visualization using the restricted quadtree triangulation. In Proc. IEEE Visualization, pp. 19–26, 1998.

[PG07] Pajarola, R., and Gobbetti, E. Survey on semi-regular multiresolution models for interactive terrain rendering. The Visual Computer, Vol. 23, No. 8, pp. 583–605, 2007.

[PS] Puget Sound elevation data set is available at http://www.cc.gatech.edu/projects/large_models/ps.html

[SW06] Schneider, J., and Westermann, R. GPU-Friendly High-Quality Terrain Rendering. Journal of WSCG, Vol. 14, pp. 49–56, 2006.

[Ulr00] Ulrich, T. Rendering massive terrains using chunked level of detail. ACM SIGGraph Course “Super-size it! Scaling up to Massive Virtual Worlds”, 2000.

[WNC87] Witten, I.H., Neal, R.M., and Cleary J.G., Arithmetic coding for data compression. Comm. ACM, Vol. 30, No. 6, pp. 520–540, June 1987.

Generalized Heat Kernel Signatures

Valentin Zobel
Zuse-Institut Berlin, Germany
zobel@zib.de

Jan Reininghaus
Zuse-Institut Berlin, Germany
reininghaus@zib.de

Ingrid Hotz
Zuse-Institut Berlin, Germany
hotz@zib.de

ABSTRACT

In this work we propose a generalization of the Heat Kernel Signature (HKS). The HKS is a point signature derived from the heat kernel of the Laplace-Beltrami operator of a surface. In the theory of exterior calculus on a Riemannian manifold, the Laplace-Beltrami operator of a surface is a special case of the Hodge Laplacian which acts on r-forms, i. e. the Hodge Laplacian on 0-forms (functions) is the Laplace-Beltrami operator. We investigate the usefulness of the heat kernel of the Hodge Laplacian on 1-forms (which can be seen as the vector Laplacian) to derive new point signatures which are invariant under isometric mappings. A similar approach used to obtain the HKS yields a symmetric tensor field of second order; for easier comparability we consider several scalar tensor invariants. Computed examples show that these new point signatures are especially interesting for surfaces with boundary.

Keywords: Shape analysis, Hodge Laplacian, heat kernel, discrete exterior calculus

1 INTRODUCTION

The identification of similarly shaped surfaces or parts of surfaces, represented as triangle meshes, is an important task in computational geometry. In this paper, we consider two surfaces as being similar if there is an isometry between them. For example, all meshes describing different poses of an animal are considered to be similar.

One approach to solve this problem makes use of spectral analysis of the Laplace-Beltrami operator Δ_0 of the surface. The Laplace-Beltrami operator Δ_0 describes diffusion processes, is by definition invariant under isometries, and is known to reveal many geometric properties of the surface.

In [8] the eigenvalues of the Laplace-Beltrami operator are proposed as a 'Shape-DNA'. If two surfaces are isometric, then the eigenvalues of the respective Laplace-Beltrami operators coincide. While one can construct counter examples to the converse of this statement, this does not seem to pose a problem in practice.

In contrast to this global characterization of surfaces, in [10] the eigenvalues and eigenfunctions of the Laplace-Beltrami operator are used to compute a point signature. This point signature is a function on the surface containing a scale parameter, and is called *Heat Kernel Signature*. For benchmarks evaluating the Heat Kernel Signature and other methods we refer the reader to [3], [4].

In this work we propose and investigate a generalization of the *Heat Kernel Signature*. The Laplace-Beltrami operator Δ_0 of a surface can be generalized to the Hodge Laplacian Δ_r which is an operator acting on r-forms. This operator is defined in the setting of exterior calculus in Section 2 and its heat kernel is introduced in Section 3. We can then derive a new isometry invariant point signature from the Hodge-Laplacian on 1-forms Δ_1 in Section 4. This yields a symmetric tensor field of second order containing a scale parameter. As it is difficult to compare and quantify such tensor fields, we consider several scalar valued tensor invariants for the purpose of surface analysis. To increase the reproducibility of the results shown in Section 6, we give some details about our implementation of this method in Section 5. For our discretization of Δ_1 we use the theory of discrete exterior calculus (DEC) which mimics the theory of exterior calculus on a discrete level.

2 MATHEMATICAL BACKGROUND

To generalize the Laplace-Beltrami operator and the heat kernel to r-forms it is beneficial to employ the theory of exterior calculus on a Riemannian manifold. We will give a short introduction to this topic in this section. An extensive introduction to exterior calculus can be found for example in the textbook [1].

For simplicity we restrict ourselves to a Riemannian manifold (M,g) of dimension 2. Readers who are not familiar with Riemannian manifolds may think of M being a surface embedded in $\mathbb{R}^3$. In this case the Riemannian metric g is given by the first fundamental form, i. e. g_p is the scalar product on the tangent space $T_p(M)$ at p which is induced by the standard scalar product on $\mathbb{R}^3$.

The set of r-forms on M is denoted by $\bigwedge^r(M)$, where $r = 0 \ldots 2$. A 0-form on M is a smooth function from M to $\mathbb{R}$, consequently $\bigwedge^0(M) = C^\infty(M)$. A 1-form on

M is a smooth map which assigns each $p \in M$ a linear map from $T_p(M)$ to $\mathbb{R}$, i. e. an element of the dual space $(T_p(M))^*$ of $T_p(M)$. A 2-form α on M is a smooth map which assigns each $p \in M$ a bilinear form on $T_p(M)$ which is skew-symmetric, that is for each $p \in M$ and $v, w \in T_p(M)$ we have $\alpha_p(v,w) = -\alpha_p(w,v)$. We will later see that a 1-form can be identified with a vector field while a 2-form can be interpreted as a function on the manifold.

The Hodge-Laplace operator will now be defined in terms of local coordinates. Let (U,ϕ) be a chart with coordinate functions (x_1,x_2), i. e. $\phi(p) = (x_1(p),x_2(p)) \in \mathbb{R}^2$. The tangent vectors to the coordinate lines which are denoted by $\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}$, or shorter ∂_1, ∂_2, form a frame on U, i. e. $(\partial_1)_p, (\partial_2)_p$ is a basis of $T_p(M)$ for each $p \in U$. The differentials dx_1, dx_2 of x_1 and x_2 form a coframe on U, i. e. $(dx_1)_p, (dx_2)_p$ is a basis of $(T_p(M))^*$, and we have $dx_i(\partial_j) = \delta^i_j$. Thus, for any 1-form $\alpha \in \bigwedge^1(M)$ there are functions $f_1, f_2 \in \mathbb{C}^\infty(U)$ such that

$$\alpha|_U = f_1\, dx_1 + f_2\, dx_2 \ ,$$

where $f_1 = \alpha(\partial_1)$, $f_2 = \alpha(\partial_2)$.

The wedge prodcut $\wedge$ of two 1-forms α, β is defined pointwise at each $p \in M$ by

$$(\alpha_p \wedge \beta_p)(v,w) = \alpha_p(v)\beta_p(w) - \beta_p(v)\alpha_p(w)$$

for all $v, w \in T_p(M)$. A two form $\alpha \in \bigwedge^2(M)$ can thereby be represented by $\alpha|_U = f\, dx_1 \wedge dx_2$, where $f = \alpha(\partial_1, \partial_2) \in C^\infty(M)$.

There is an isomorphism between vector fields and 1-forms on M which is called flat operator and denoted by $^\flat$. For a vector field v it is defined by $v^\flat_p(\cdot) = g(v_p, \cdot)$ at each $p \in M$. Its inverse is the sharp operator $^\sharp$. If e_1, e_2 is an orthonormal basis of $T_p(M)$ and $\varepsilon_1, \varepsilon_2$ its dual basis we have $(\lambda_1 e_1 + \lambda_2 e_2)^\flat = \lambda_1 \varepsilon_1 + \lambda_2 \varepsilon_2$, where $\lambda_1, \lambda_2 \in \mathbb{R}$.

The differential d takes a function f on M to the 1-form

$$d_0 f = \frac{\partial f}{\partial x_1} dx_1 = \frac{\partial f}{\partial x_2} dx_2 \ ,$$

i. e. d_0 maps 0-forms to 1-forms. One may think of d_0 as ∇. We will denote d also by d_0 and define the map d_1 taking 1-forms to 2-forms by

$$d_1(f_1\, dx_1 + f_2\, dx_2) = \left(\frac{\partial f_2}{\partial x_1} - \frac{\partial f_1}{\partial x_2}\right) dx_1 \wedge dx_2 \ .$$

d_1 can be interpreted as $\nabla\times$. The maps d_0 and d_1 are referred to as *exterior derivative*.

Next we will define the maps δ_1 and δ_2 which take 1-forms to 0-forms and 2-forms to 1-forms, respectively, and are also called *codifferential*. These maps depend, in contrast to d_0 and d_1, on the metric of M. We set $g_{ij} = g\left(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial x_j}\right)$ and $G = \sqrt{\det[g_{ij}]}$. For simplicity we use orthogonal coordinates, that is $[g_{ij}]$ is a diagonal matrix. This is not a restriction, since any point $p \in M$ is contained in a chart with this property. The *Hodge star operator* $*_r$ is a map taking r-forms to $(2-r)$-forms, $r = 0, \ldots, 2$, defined by

$$\begin{aligned} *_0 f &= G f\, dx \wedge dy \ , \\ *_1(f_1\, dx_1 + f_2\, dx_1) &= -g_{22} G f_2\, dx_1 + g_{11} G f_1\, dx_2 \ , \\ *_2(f dx_1 \wedge dx_2) &= \frac{f}{G} \ . \end{aligned}$$

Now δ_1 and δ_2 are defined by

$$\delta_1 = -*_2 d_1 *_1 \ , \quad \delta_2 = -*_1 d_0 *_2 \ ,$$

which can be rewritten to

$$\begin{aligned} \delta_1(f_1\, dx_1 + f_2\, dx_2) &= -\frac{1}{G}\left(\frac{\partial g_{11} G f_1}{\partial x_1} + \frac{\partial g_{22} G f_2}{\partial x_2}\right), \\ \delta_2(f\, dx_1 \wedge dx_2) &= g_{22} G \frac{\partial \frac{f}{G}}{\partial x_2} dx_1 - g_{11} G \frac{\partial \frac{f}{G}}{\partial x_1} dx_2 \ . \end{aligned}$$

One may think of $-\delta_1$ as $\nabla\cdot$ and $-\delta_1$ as $\nabla^\perp$.

The *Hodge Laplacian* $\Delta_r : \bigwedge^r(M) \to \bigwedge^r(M)$, where $r = 0, \ldots, 2$, is now defined by

$$\begin{aligned} \Delta_0 &= \delta_1 d_0 \ , \\ \Delta_1 &= \delta_2 d_1 + d_0 \delta_1 \ , \\ \Delta_2 &= d_1 \delta_2 \ . \end{aligned}$$

Sometimes Δ_r is also called Laplace-de Rham operator or just Laplacian, where Δ_0 is also referred to as Laplace-Beltrami operator. If $M = \mathbb{R}^2$ with standard coordinates we have $g_{11} = g_{22} = G = 1$, thus $-\Delta_0$ coincides with the well-known definition of the Laplacian on $\mathbb{R}^2$, i. e. $\Delta_0 = \frac{\partial^2}{\partial^2 x_1} + \frac{\partial^2}{\partial^2 x_2}$.

3 HEAT KERNEL

The basic properties of heat diffusion on a Riemannian manifold will be introduced in this section. Of special interest for us is the heat kernel and its generalization to 1-forms. In Section 4 we will derive point signatures from the heat kernel for 1-forms in a similar way as the Heat Kernel Signature is derived from the heat kernel for functions. For details on the heat kernel for r-forms see [9].

Let (M,g) be a 2-dimensional, compact, oriented Riemannian manifold. Given an initial heat distribution $f(p) = f(0,p) \in C^\infty(M)$ on M, considered to be perfectly insulated, the heat distribution $f(t,p) \in C^\infty(M)$ at time t is governed by the *heat equation*

$$(\partial_t + \Delta_0) f(t,p) = 0 \ .$$

The function $k^0(t,p,q) \in C^\infty(\mathbb{R}^+ \times M \times M)$ such that for all $f \in C^\infty(M)$

$$\begin{aligned} (\partial_t + (\Delta_0)_p) k^0(t,p,q) &= 0 \ , \\ \lim_{t\to 0} \int k^0(t,p,q) f(q)\, dq &= f(p) \ , \end{aligned}$$

is called *heat kernel*. $(\Delta_0)_p$ denotes the Laplacian acting in the p variable. Using the heat kernel one can define the *heat operator* H_t for $t > 0$ by

$$H_t f(p) = \int_M k^0(t,p,q) f(q)\, dq \ .$$

One can show that $f(t,p) = H_t f(p)$ solves the Heat equation, thus H_t maps an initial heat distribution to the heat distribution at time t. The heat kernel can be computed from the eigenvalues λ_i and the corresponding eigenfunctions ϕ_i of Δ_0 by the formula

$$k^0(t,p,q) = \sum_i e^{-\lambda_i t} \phi_i(p)\phi_i(q) \ .$$

Next we will generalize the heat kernel to 1-forms which results in a so-called double 1-form. A double 1-form is a smooth map which assigns each $(p,q) \in M \times M$ a bilinear map $T_pM \times T_qM \to \mathbb{R}$. Consequently, if β is a double form on M, $v \in T_p(M)$, $w \in T_q(M)$, then $q \mapsto \beta(p,q)(v,\cdot)$ and $p \mapsto \beta(p,q)(\cdot,w)$ are 1-forms on M. The heat kernel for 1-forms is now a double form $k^1(t,p,q)$ depending smoothly on an additional parameter t, which satisfies for each $\alpha \in \bigwedge^k(M)$

$$(\partial_t + (\Delta_1)_p) k^1(t,p,q) = 0 \ ,$$
$$\lim_{t\to 0} \int_M k^1(t,p,q) \left(\cdot, \alpha^\sharp(q)\right) dq = \alpha(p)(\cdot) \ .$$

Note that, given $\alpha \in \bigwedge^1(M)$ and $p,q \in M$ we obtain a bilinear map $T_p(M) \times T_q(M) \to \mathbb{R}$ by multiplying $\alpha(p)$ and $\alpha(q)$; thus

$$(p,q) \mapsto \alpha(p)(\cdot)\, \alpha(q)(\cdot)$$

is a double form. Similarly to the heat kernel for functions, we can compute the heat kernel for 1-forms from the eigenvalues λ_i and the eigenforms α_i of Δ_1 by

$$k^1(t,p,q)(\cdot,\cdot) = \sum_i e^{-\lambda_i t} \alpha_i(p)(\cdot)\, \alpha_i(q)(\cdot) \ .$$

4 POINT SIGNATURES FROM THE HEAT KERNEL FOR 1-FORMS

In this section we will derive new point signatures from the heat kernel of 1-forms. This is done in a similar way as the Heat Kernel Signature is derived from the heat kernel for functions (0-forms). The main difference is that this approach does not result in a time-dependent function for the heat kernel of 1-forms, instead we obtain a time-dependent tensor field. Thus, to obtain comparable values, we consider scalar tensor invariants. In this way we obtain several point signatures which are especially interesting for manifolds with boundary, as we will see in Section 6.

The Heat Kernel Signature at p is defined by

$$t \mapsto k^0(t,p,p) \ ,$$

i. e. a function $\mathbb{R}^+ \to \mathbb{R}$ is assigned to each point $p \in M$. It is shown in [10] that two points p,q have similar shaped neighborhoods if $\{k(t,p,p)\}_{t>0}$ and $\{k(t,q,q)\}_{t>0}$ coincide.

The analogous definition for the heat kernel for 1-forms,

$$t \mapsto k^1(t,p,p) \ ,$$

assigns each point $p \in M$ a bilinear form on $T_p(M)$ or equivalently a symmetric covariant tensor of second order. Comparing covariant tensors of second order on $T_p(M)$ and $T_q(M)$ is not possible unless we have a meaningful map between $T_p(M)$ and $T_q(M)$. It is therefore difficult to compare $\{k^1(t,p,p)\}_{t>0}$ and $\{k^1(t,q,q)\}_{t>0}$ directly. However, we can consider scalar tensor invariants which are independent of the chosen orthonormal basis of the tangent space.

If e_1, e_2 is an orthonormal basis of $T_p(M)$ we can assign to each bilinear form β a matrix $B = (b_{ij})$, where $b_{ij} = \beta(e_i, e_j)$, $i,j = 1,2$. Now B is the matrix representation of β with respect to the orthonormal basis e_1, e_2 and the eigenvalues of β are defined to be the eigenvalues of B. If $\tilde{e}_1, \tilde{e}_2$ is another orthonormal basis and S the orthogonal matrix satisfying $\tilde{e}_1 = Se_1, \tilde{e}_2 = Se_2$, then the corresponding matrix representation $\tilde{B}$ of α is given by $\tilde{B} = SBS^T$, and with that the definition of the eigenvalues of β is independent of a certain orthonormal basis. Consequently, if λ_1 is the larger and λ_2 the smaller eigenvalue of β, quantities like λ_1 or λ_2 or combinations of it like the trace $\mathrm{tr}(\beta) = \mathrm{tr}(B) = \lambda_1 + \lambda_2$ or the determinant $\det(\beta) = \det(B) = \lambda_1\lambda_2$ are scalar tensor invariants. Using such tensor invariants we obtain point signatures like $\{\mathrm{tr}(k^1(t,p,p))\}_{t>0}$ which can be compared similarly as the Heat Kernel Signature, see [10] for details.

5 NUMERICAL REALIZATION

To compute our point signatures we need a matrix representation of the bilinear forms $k^1(t,p,p)$. We will use the equation

$$k^1(t,p,p)(\cdot,\cdot) = \sum_i e^{-\lambda_i t} \alpha_i(p)(\cdot)\, \alpha_i(p)(\cdot) \ , \quad (1)$$

where λ_i and α_i are the eigenvalues and eigenforms of Δ_1. For the computation of the eigenvalues and eigenforms we use the theory of discrete exterior calculus (DEC), which mimics the theory of exterior calculus on surfaces approximated as triangle meshes. A short introduction to DEC is given in Subsection 5.1.

Unfortunately the computation of the eigenvalues and eigenforms of Δ_1 using DEC is not straightforward. The common definitions work only for very special triangulations. We propose a solution to this problem in Subsection 5.2. Moreover we explain a way to realize the product $\alpha_i(p)(\cdot)\, \alpha_i(p)(\cdot)$ of two eigenforms which is not obvious for discrete r-forms.

5.1 Discrete Exterior Calculus

DEC deals with discrete forms which are defined on on a triangle mesh as an approximation of a surface. Additionally counterparts of operators like the exterior derivative and the Hodge star operator are defined for discrete forms. This enables us to define a discrete Hodge Laplacian. Thus DEC mimics the theory of smooth r-forms on surfaces. For details on DEC we refer the reader to [7], which is the most extensive source, as well as to [5] and [6].

Let K be a triangle mesh with vertex set $V = \{v_i\}$, edge set $E = \{e_i\}$ and triangle set $T = \{t_i\}$. We assume that all triangles and edges have a fixed orientation. The orientation of a vertex is always positive; the orientation of an edge e_i is given by an order of vertices $e = [v_i v_j]$; the orientation of a triangle t is given by a cyclic order of vertices $t = [v_i v_j v_k]$. If v is a vertex of the edge $e = [v_i v_j]$, the orientations of v and e are said to agree if $v = v_j$ and disagree if $v = v_i$. Similarly, given an edge e of a triangle t, the orientations of e and t are said to agree (disagree) if the vertices of e occur in the same (opposite) order in t.

Discrete 0-forms, 1-forms and 2-forms are defined to be functions from V, E and T to $\mathbb{R}$, respectively. The function values should be understood as the integral of a continuous 0-form, 1-form or 2-form over a vertex, edge or triangle, respectively. Note that reversing the orientation of vertices, edges or triangles changes the sign of the associated integral values, thus the same holds for discrete r-forms. Of course, this definition of discrete r-forms does not allow a point-wise evaluation.

However, it is possible to interpolate discrete r-forms by Whitney forms which are piecewise linear r-forms on the triangles. Whitney 0-forms are the so-called hat functions, i. e. ϕ_{v_i} is the piecewise linear function with $\phi_{v_i}(v_j) = \delta^i_j$. For an edge $e = [v_i, v_j]$ the Whitney 1-form ϕ_e is supported on the triangles adjacent to e and given by $\phi_e = \phi_{v_i}\, d\phi_{v_j} - \phi_{v_j}\, d\phi_{v_i}$. Note that ϕ_e is piecewise linear on each triangle, but discontinuous on the edge. However, the integral of both parts of ϕ_e over e is 1. We also have that the integral of ϕ_e is 0 over each edge different from e. There is a similar definition for Whitney 2-forms which we omit here. The Whitney interpolant $\mathscr{I}\alpha$ of a discrete 0-form α is now given by

$$\mathscr{I}\alpha = \sum_{i=1,\dots,|V|} \alpha(v_i)\phi_{v_i} \ .$$

The Whitney interpolant for discrete 1-forms and 2-forms is defined analogously.

0-forms, 1-forms and 2-forms can be seen as vectors in $\mathbb{R}^{|V|}$, $\mathbb{R}^{|E|}$ and $\mathbb{R}^{|T|}$. Thus operators like the exterior derivative, the hodge star operator and the codifferential are defined as matrices. To define the discrete exterior derivate we need to define the boundary operator first. The boundary operator ∂_1 is given by the matrix of dimension $|V| \times |E|$ with the entries

$$(\partial_1)_{ij} = \begin{cases} 1 \ , & \text{orientations of } v_i \text{ and } e_j \text{ agree} \ , \\ -1 \ , & \text{orientations of } v_i \text{ and } e_j \text{ disagree} \ , \end{cases}$$

if v_i is a vertex of the edge e_j and zero otherwise. The boundary operator ∂_2 is now defined analogously by

$$(\partial_1)_{ij} = \begin{cases} 1 \ , & \text{orientations of } e_i \text{ and } t_j \text{ agree} \ , \\ -1 \ , & \text{orientations of } e_i \text{ and } t_j \text{ disagree} \ , \end{cases}$$

if the e_j is an edge of the triangle t_j and zero otherwise. The discrete exterior derivate is now defined to be the transpose of the boundary operator, i. e.

$$d_0 = (\partial_1)^T \quad , \quad d_1 = (\partial_2)^T \ .$$

Thus, as for smooth r-forms we have that d_0 maps 0-forms to 1-forms, and d_1 maps 1-forms to 2-forms.

While the hodge star operator $*_r$ in the continuous case maps r-forms to $(2-r)$-forms, the discrete hodge star operator maps a discrete r-form to a so-called dual $(2-r)$-form which is defined on the dual mesh. We assume for the moment that every triangle $t \in T$ contains its circumcenter. Then the (circumcentric) dual mesh is a cell decomposition of K where the cells are constructed as follows: The dual 0-cell $\star t$ of a triangle $t \in T$ is the circumcenter of t. The dual 1-cell $\star e$ of an edge $e \in E$ consists of the two line segments connecting the circumcenters of the triangles adjacent to e and the midpoint of e. The dual 2-cell $\star v$ of a vertex $v \in V$ is the area around v which is bounded by the dual 1-cells of the edges adjacent to v. Note that the dual mesh coincides with the Voronoi tesselation of K corresponding to the vertex set V, see [2] for details.

A dual r-form is now a map which assigns each dual r-cell a real number. Thus dual 0-forms, 1-forms and 2-forms can be represented as vectors in $\mathbb{R}^{|T|}$, $\mathbb{R}^{|E|}$ and $\mathbb{R}^{|V|}$. The exterior derivative on dual 0-forms and dual 1-forms is defined by the matrices

$$d_0^{Dual} = d_1^T = \partial_2 \ , \quad d_1^{Dual} = -d_0^T = -\partial_1 \ .$$

The discrete Hodge star operator $*_r$ which maps r-forms to dual $2-r$ forms is given by square matrices

$$*_0 \in \mathbb{R}^{|V|\times|V|} \ , \quad *_1 \in \mathbb{R}^{|E|\times|E|} \ , \quad *_2 \in \mathbb{R}^{|T|\times|T|} \ .$$

Unfortunately there is no unique way to define the entries of these matrices. A possible choice for $*_0$, $*_1$ and $*_2$ are diagonal matrices with entries given by

$$(*_0)_{ii} = \frac{|\star v_i|}{|v_i|} \ , \quad (*_1)_{ii} = \frac{|\star e_i|}{|e_i|} \ , \quad (*_2)_{ii} = \frac{|\star t_i|}{|t_i|} \ ,$$

where $|v| = 1$, $|e|$ is the length of e, $|t|$ is the area of t and analogously for dual cells. Since this is the common

definition in DEC, see [7] and [5] for example, we also denote this Hodge star by $*_r^{DEC}$.

Another possible definition, suggested in [6], is to define $(*_0)_{ij}$ as the the L^2-inner product of the Whitney 0-forms ϕ_{v_i} and ϕ_{v_j}, and analogously for $*_1$ and $*_2$ using Whitney 1-forms and 2-forms corresponding to edges and triangles, respectively. For more details and an explicit computation of the entries of $*_r^{Whit}$ we refer to [11]. We denote this Hodge star operator also by $*_r^{Whit}$ in allusion to the use of Whitney forms. The advantages and disadvantages of $*^{DEC}$ and $*^{Whit}$ in view of spectral analysis of the Hodge Laplacian will be discussed in Subsection 5.2.

To map dual $(2-r)$-forms to discrete r-forms we need an inverse Hodge star operator $*_{2-r}^{Dual}$. An obvious choice would be $*^{-1}$ but in this case the property $*_r *_{2-r} \alpha = (-1)^{r(2-r)}\alpha$ which we have for a smooth r-form α would not hold. Instead $*_{2-r}^{Dual}$ is defined by

$$*_{2-r}^{Dual} = (-1)^{r(2-r)}(*_r)^{-1} \quad .$$

Now, similarly as for smooth r-forms, we define the discrete codifferential which maps discrete r-forms to discrete $(r-1)$-forms for $r = 1,2$ by

$$\begin{aligned} \delta_1 &= -*_2^{Dual} d_1^{Dual} *_1 \quad , \\ \delta_2 &= -*_1^{Dual} d_0^{Dual} *_2 \quad . \end{aligned}$$

This enables us to define the discrete Hodge Laplacian Δ_r just the same way as in the smooth case by

$$\begin{aligned} \Delta_0 &= \delta_1 d_0 \quad , \\ \Delta_1 &= \delta_2 d_1 + d_0 \delta_1 \quad , \\ \Delta_2 &= d_1 \delta_2 \quad . \end{aligned}$$

Thus Δ_r can be assembled from the boundary operator and the discrete Hodge star operator by

$$\begin{aligned} \Delta_0 &= *_0^{-1} \partial_1 *_1 \partial_1^T \quad , \\ \Delta_1 &= *_1^{-1} \partial_2 *_2 \partial_2^T + \partial_1^T *_0^{-1} \partial_1 *_1 \quad , \\ \Delta_2 &= \partial_2^T *_1^{-1} \partial_2 *_2 \quad . \end{aligned}$$

5.2 Numerical Computation of the Point Signatures

To compute $k^1(t,p,p)$ using the formula (1) we need to compute the eigenvalues and eigenforms of Δ_1 in a first step. We will see that we need certain combinations of the Hodge star operators $*_r^{DEC}$ and $*_r^{Whit}$ to accomplish this. In a second step we need to compute the products of two eigenforms $\alpha_i(p)(\cdot)\,\alpha_i(p)(\cdot)$. Since DEC does not provide such a product, we use Whitney forms to interpolate smooth r-forms from discrete r-forms. This results in matrix representations of $\alpha_i(p)(\cdot)\,\alpha_i(p)(\cdot)$ which can be summed easily.

To compute the eigenvalues of Δ_1 we need to solve the eigenvalue problem

$$\Delta_1 \alpha = \left(*_1^{-1} \partial_2 *_2 \partial_2^T + \partial_1^T *_0^{-1} \partial_1 *_1\right) \alpha = \lambda \alpha \quad ,$$

or alternatively the generalized eigenvalue problem

$$\left(\partial_2 *_2 \partial_2^T + *_1 \partial_1^T *_0^{-1} \partial_1 *_1\right) \alpha = \lambda *_1 \alpha \quad .$$

The advantage of the generalized eigenvalue problem is that one does not need the inverse of $*_1$, but only needs the inverse of $*_0$. However, to solve such a generalized eigenvalue problem with usual numerical methods, e. g. by using the command `eigs` in Matlab, the matrix on the right hand side, i. e. $*_1$, must be symmetric positive definite. Moreover we need to compute the inverse of $*_0$. So, which of the matrices $*_r^{DEC}$, $*_r^{Whit}$, $r = 0,\ldots,2$, are invertible, which are also symmetric positive definite?

Since $*_1^{DEC}$ is a diagonal matrix with diagonal entries given by

$$(*_1)_{ii} = \frac{|\star e_i|}{|e_i|} \quad ,$$

it is invertible if and only if $|\star e_i|/|e_i| \neq 0$ for $i = 1,\ldots,|E|$; if $|\star e_i|/|e_i| > 0$ for $i = 1,\ldots,|E|$ it is also positive definite. The length $|e|$ of an edge is obviously always positive. For the length $|\star e|$ of the dual 1-cell of an edge e this is possibly not the case. Of course, if we assume that the circumcenter of each $t \in T$ is contained in t, as in the previous section, the length of $\star e$ is the sum of the lengths of the two line segments connecting the circumcenters of the two triangles adjacent to e with the midpoint of e and thus positive. But this is not a viable assumption in applications. One can solve this problem in the following way: Let t be a triangle adjacent to e. If t and the circumcenter of t lie on different sides of the line containing e, then the according line segment counts negative. Thus the length $|\star e|$ of a dual 1-cell $\star e$ can be negative; this is the case if and only if this edge violates the local Delaunay property and consequently the entries of $*^{DEC}$ are only nonnegative if K is an (intrinsic) Delaunay triangulation, see [2] for details on Delaunay triangulations of triangle meshes. Since it is a very strong condition to assume that K is a Delaunay triangulation and moreover not sufficient for positive definiteness of $*^{DEC}$, only positive semidefiniteness, we cannot assume that $*_1^{DEC}$ is invertible or even positive definite.

Similarly $*_0^{DEC}$ is positive definite if $|\star v_i| > 0$ for $i = 1,\ldots,|V|$. The computation of the area $|\star v|$ of a dual 2-cell $\star v$ is shown in Figure 1, for details we refer the reader to [11]. Note that $|\star v|$ can be positive even if K is not a Delaunay triangulation; $|\star v|$ is only negative for rather degenerate meshes. Thus we can assume that $*_0^{DEC}$ is positive definite and thus invertible. Finally, $*_2^{DEC}$ is obviously positive definite.

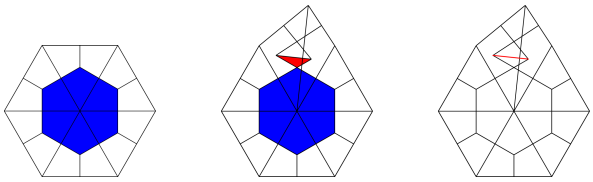


Figure 1: Primal and dual meshes. The left mesh is Delaunay, whereas the other meshes are not Delaunay. The middle mesh shows a dual 0-cell whose area is given by the blue area minus the red area. The red line in the right mesh shows a dual 1-cell with negative length.

The positive definiteness of $*_r^{Whit}$ follows from the fact that $\alpha^T *_r^{Whit} \beta$ is the L^2-inner product of the Whitney interpolants $\mathscr{I}\alpha$ and $\mathscr{I}\beta$ of two discrete r-forms α, β, thus

$$\alpha^T *_r^{Whit} \alpha > 0$$

for any r-form $\alpha \neq 0$. Consequently $*_r^{Whit}$ is also invertible, but unfortunately we cannot use the inverse of $*_r^{Whit}$. The reason for this is that $*_k^{Whit}$ is not diagonal (unless $r = 2$) and thus the inverse is in general not a sparse matrix which is a mandatory condition for large meshes.

As a consequence, to solve the generalized eigenvalue problem for Δ_1, we have to use $(*_0^{DEC})^{-1}$ and $*_1^{Whit}$ on the right hand side. For $*_1$ on the left hand side we can choose either $*_1^{DEC}$ or $*_1^{Whit}$, both work properly as the numerical tests in [11] show. For $*_2$ there is nothing to choose, since $*_2^{DEC} = *_2^{Whit}$.

We now discuss the computation of the matrix representation of $k^1(t,p,p)$ from the eigenvalues and eigenforms of Δ_1 using the formula

$$k^1(t,p,p)(\cdot,\cdot) = \sum_i e^{-\lambda_i t}\, \alpha_i(p)(\cdot)\, \alpha_i(p)(\cdot) \ .$$

One difficulty is to compute the product of the eigenforms α_i of Δ_1. The α_i are only available as discrete 1-forms, but unfortunately DEC does not provide such a product. To overcome this problem we interpolate the discrete 1-forms using Whitney forms. The resulting smooth forms can be multiplied easily. Though, as noted in the previous subsection, the Whitney forms are only continuous within the triangles, thus it is not possible to evaluate the resulting tensors on the vertices. Instead, we evaluate the tensors on the barycenters of the triangles.

We proceed with a detailed description of the computation of the matrix representation of $k^1(t,p,p)$. Let $t = [v_i v_j v_k]$ be a triangle, while the orientation of the edges is given by $e_i = [v_j v_k]$, $e_j = [v_k v_i]$ and $e_k = [v_i v_j]$. Using the orthonormal basis

$$e_1 = \frac{v_j - v_i}{\|v_j - v_i\|} \quad , \quad e_2 = \frac{(v_k - v_i) - \langle v_k - v_i, e_1\rangle e_1}{\|(v_k - v_i) - \langle v_k - v_i, e_1\rangle e_1\|}$$

and choosing v_i as origin we obtain

$$v_i = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \ , \quad v_j = \begin{pmatrix} x_j \\ 0 \end{pmatrix} \ , \quad v_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} \ ,$$

where $x_j = \langle v_j, e_1\rangle$, $x_k = \langle v_k, e_1\rangle$, $y_k = \langle v_k, e_2\rangle$. Now easy calculations show for the hat functions $\phi_{v_i}, \phi_{v_j}, \phi_{v_k}$ that

$$(d\phi_i)^\sharp = \begin{pmatrix} -\frac{1}{x_j} \\ \frac{x_k}{x_j y_k} - \frac{1}{y_k} \end{pmatrix} \ ,$$

$$(d\phi_j)^\sharp = \begin{pmatrix} \frac{1}{x_j} \\ -\frac{x_k}{x_j y_k} \end{pmatrix} \ ,$$

$$(d\phi_k)^\sharp = \begin{pmatrix} 0 \\ \frac{1}{y_k} \end{pmatrix} \ ,$$

where we used the sharp operator to identify 1-forms with vectorfields. Let now α be an eigenform of Δ_1, then the Whitney interpolant $\mathscr{I}\beta$ at the barycenter p of T is given by

$$(\mathscr{I}\alpha)(p) = \frac{1}{3}(\alpha(e_k)(d\phi_{v_j} - d\phi_{v_i}) + \alpha(e_i)(d\phi_{v_k} - d\phi_{v_j}) + \alpha(e_{v_j})(d\phi_{v_i} - d\phi_{v_k})) \ .$$

The matrix representaion of $\mathscr{I}\alpha(p)(\cdot)\,\mathscr{I}\alpha(p)(\cdot)$ is now given by

$$\Big((\mathscr{I}\alpha)^\sharp(p)\Big)\Big((\mathscr{I}\alpha)^\sharp(p)\Big)^T \ ,$$

and the matrix representation of $k^1(t,p,p)$ by

$$\sum_i e^{-\lambda_i t}\Big((\mathscr{I}\alpha_i)^\sharp(p)\Big)\Big((\mathscr{I}\alpha_i)^\sharp(p)\Big)^T \ . \qquad (2)$$

6 RESULTS

In this section we visualize our point signatures with colormaps; small values are represented by blue and high values by red. The surfaces we investigate are the trim-star model, the armadillo model and the Caesar model, provided by the AIM@SHAPE Shape Repository, a surface representing a mandible produced by M. Zinser, Universitätsklinik Köln, and a square. Plots of the point signatures for these surfaces are given for different time values and compared with the Heat Kernel Signature.

We approximate the sum in equation 2 by the first 100 summands, i. e. we have to compute the 100 smallest eigenvalues and the corresponding eigenvectors of Δ_1. The number of summands needed depends on the surface. In our examples more summands show no significant improvement. The computation of the eigenvalues and eigenvectors of Δ_1, for which we use Matlab, needs most time, everything else can be done interactively. Timings are shown in Table 1; for comparison we also give timings for the computation of 100

Model	Vertices	Δ_1	Δ_0
Mandible	11495	39.9	8.9
Trim-star	5192	17.2	7.6
Square	4096	13.4	3.4
Caesar	4717	15.0	3.0

Table 1: Timings in seconds for the computation of 100 eigenvalues and eigenvectors of Δ_1 and Δ_0.

eigenvalues and eigenvectors of Δ_0, which are needed to compute the HKS.

To avoid readjusting the colormap for different values of t we plot the function

$$\frac{\operatorname{tr}\left(k^1(t,p,p)\right)}{\int_M \operatorname{tr}\left(k^1(t,p,p)\right)\,dp} \quad ,$$

rather than $\operatorname{tr}\left(k^1(t,p,p)\right)$, and analogously for other invariants. Such a normalization is also used in [10] to ensure that different values of t contribute approximately equally when comparing two signatures.

In the case of a closed surface the smaller and the larger eigenvalue of $k^1(t,p,p)$ have very similar values for all $p \in M$ and all $t > 0$. The behavior of $\operatorname{tr}\left(k^1(t,p,p)\right)$ and $\det\left(k^1(t,p,p)\right)$ corresponds to this observation. Thus, whichever invariant we use, we obtain nearly the same information from the resulting point signature. A comparison of $\operatorname{tr}\left(k^1(t,p,p)\right)$ and the Heat Kernel Signature is shown in Figures 2 and 3. Despite the fact that the Heat Kernel Signature has high values where $\operatorname{tr}\left(e^1(t,p,p)\right)$ has low values and vice versa, both point signatures show a similar behavior for small values of t. In contrast, for large values of t their behavior is very different.

We should note here that Δ_0 has a single zero eigenvalue and the corresponding eigenfunction is constant. Thus we have

$$\lim_{t\to\infty} k^0(t,p,p) = \lim_{t\to\infty} \sum_i e^{-\lambda_i t}\phi_i(p)\phi_i(p) = \phi_0^2(p) \quad ,$$

i. e. the Heat Kernel Signature converges to a constant function which is different to zero. In contrast, Δ_1 has $2g$ eigenforms to the eigenvalue zero, where g is the genus of the surface. Now the limit

$$\lim_{t\to\infty} k^1(t,p,q)(\cdot,\cdot) = \lim_{t\to\infty} \sum_i e^{-\lambda_i t}\,\alpha_i(p)(\cdot)\,\alpha_i(p)(\cdot)$$

is zero for surfaces with $g = 0$ and nonzero for surfaces with $g > 0$.

Thus, for the mandible model in Figure 2 $\operatorname{tr}\left(k^1(t,p,p)\right)$ converges to zero, while it does not converge to zero for the trim-star in Figure 3. However, as a consequence of our normalization, the limit zero is not visible in Figure 2, we rather see how $\operatorname{tr}\left(k^1(t,p,p)\right)$ approaches zero.

To demonstrate the isometry invariance of $k^1(t,p,p)$ Figure 4 shows $\operatorname{tr}\left(k^1(t,p,p)\right)$ for different poses of the armadillo modell.

In contrast to closed surfaces the smaller and the larger eigenvalue of $k^1(t,p,p)$ behave differently for surfaces with boundary. Consequently we also have a different behavior of $\operatorname{tr}\left(k^1(t,p,p)\right)$ and $\det\left(k^1(t,p,p)\right)$, see Figure 5 for a square and Figure 6 for a model of the head of Julius Caesar. While $\operatorname{tr}\left(k^1(t,p,p)\right)$ and the Heat Kernel Signature show a similar behavior for small t in the case of a closed surface, for surfaces with boundary this is only true away from the boundary, see again Figures 5 and 6. The Heat Kernel Signature seems to be much more influenced by the boundary as $\operatorname{tr}\left(k^1(t,p,p)\right)$. We should note here that we used for the computation of the Heat Kernel Signature eigenfunctions satisfying Neumann boundary conditions, i. e. for any eigenfunction ϕ we have

$$\frac{\partial\phi}{\partial n}(p) = 0 \quad , \quad p \in \partial M \quad ,$$

where ∂M denotes the boundary of M and n denotes the normal to the boundary. If we would use Dirichlet boundary conditions instead, i. e.

$$\phi(p) = 0 \quad , \quad p \in \partial M \quad ,$$

the influence of the boundary to the Heat Kernel Signature would be even bigger.

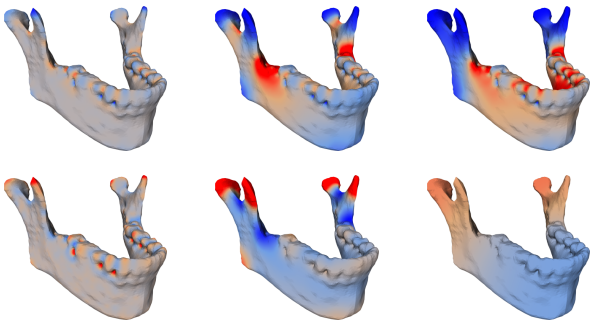


Figure 2: $\operatorname{tr}(k^1(t,p,p))$ (top) and Heat Kernel Signature (bottom) for increasing values of t.

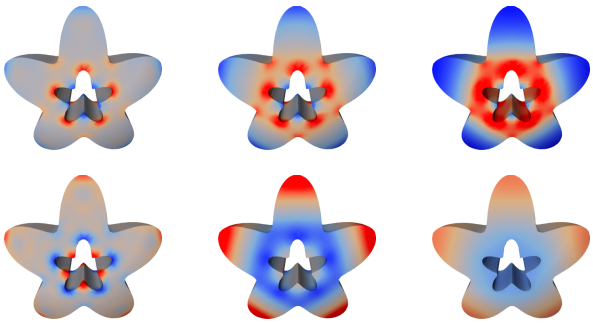


Figure 3: $\operatorname{tr}(k^1(t,p,p))$ (top) and Heat Kernel Signature (bottom) for increasing values of t.

7 CONCLUSION

In this work we derived new point signatures from the heat kernel for 1-forms. We imitated the way in which

Figure 4: $\mathrm{tr}(k^1(t,p,p))$ of the armadillo modell in different poses.

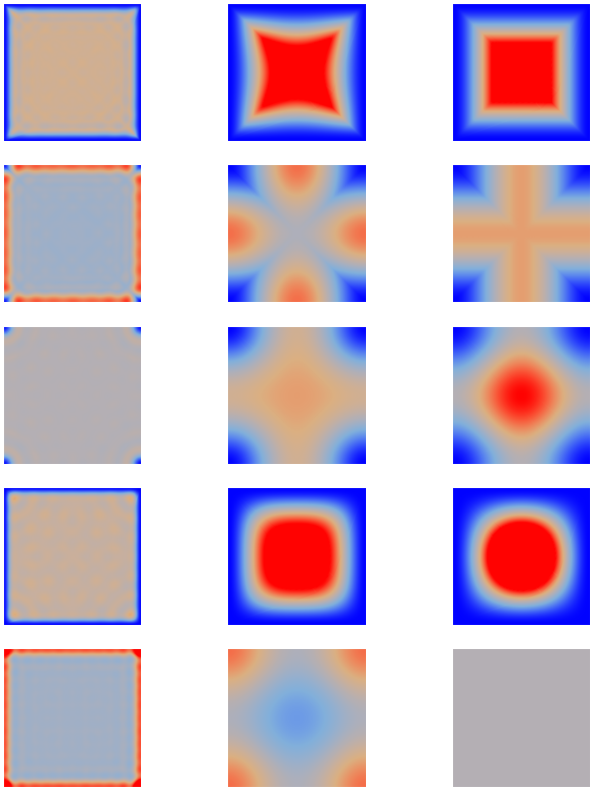


Figure 5: from top to bottom: smaller eigenvalue of $k^1(t,p,p)$, larger eigenvalue of $k^1(t,p,p)$, $\mathrm{tr}(k^1(t,p,p))$, $\det(k^1(t,p,p))$ and Heat Kernel signature for increasing values of t.

the Heat Kernel Signature is derived from the Heat Kernel of 0-forms. Since this yields a time-dependent tensor field of second order, we obtain several point signatures by considering tensor invariants like the eigenvalues, the trace and the determinant. In the case of surfaces without boundary both eigenvalues have very similar values; the trace and the determinant behave accordingly. For small time values the behavior of both eigenvalues is quite similar to the Heat Kernel Signature, but it differs for large time values. In contrast to this, the behavior of the eigenvalues is very different for surfaces with boundary, even for small time values. Thus all considered tensor invariants differ significantly from the Heat Kernel Signature. This property might bring improvements for the analysis of surfaces with boundary, compared to the Heat Kernel Signature with

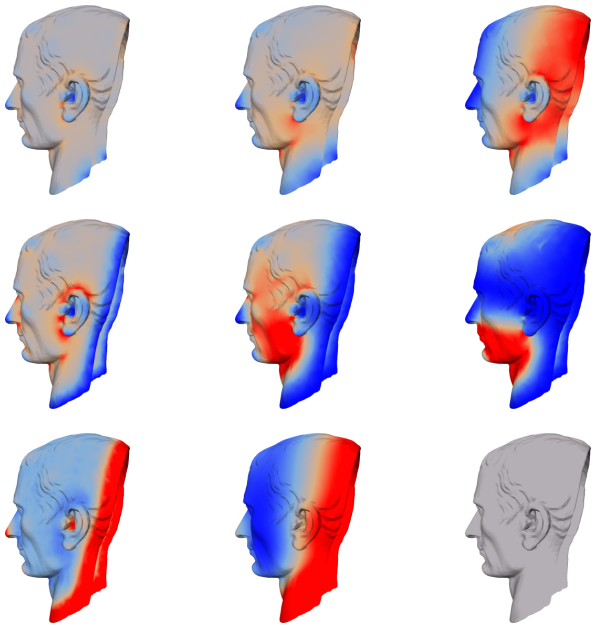


Figure 6: from top to bottom: $\mathrm{tr}(k^1(t,p,p))$, $\det(k^1(t,p,p))$ and Heat Kernel Signature for increasing values of t.

Dirichlet or Neumann boundary conditions; a further examination is left for future work.

REFERENCES

[1] R. Abraham, J.E. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Addison-Wesley, 1983.

[2] A.I. Bobenko and B.A. Springborn. A discrete laplace–beltrami operator for simplicial surfaces. *Discrete and Computational Geometry*, 38(4):740–756, 2007.

[3] A.M. Bronstein, M.M. Bronstein, B. Bustos, U. Castellani, M. Crisani, B. Falcidieno, L.J. Guibas, I. Kokkinos, V. Murino, M. Ovsjanikov, et al. SHREC 2010: robust feature detection and description benchmark. *Proc. 3DOR*, 2010.

[4] A.M. Bronstein, M.M. Bronstein, U. Castellani, B. Falcidieno, A. Fusiello, A. Godil, L.J. Guibas, I. Kokkinos, Z. Lian, M. Ovsjanikov, et al. SHREC 2010: robust large-scale shape retrieval benchmark. In *Eurographics Workshop on 3D Object Retrieval, To appear*, 2010.

[5] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 39–54, New York, NY, USA, 2006. ACM.

[6] A. Gillette. Notes on discrete exterior calculus. 2009.

[7] A.N. Hirani. *Discrete exterior calculus*. PhD thesis, Citeseer, 2003.

[8] M. Reuter, F.E. Wolter, and N. Peinecke. Laplace–beltrami spectra as 'shape-dna'of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.

[9] S. Rosenberg. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. Cambridge Univ Pr, 1997.

[10] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Proc. Eurographics Symposium on Geometry Processing (SGP)*, 2009.

[11] Valentin Zobel. *Spectral Analysis of the Hodge Laplacian on Discrete Manifolds*. Master Thesis, 2010.

Plausible and Realtime Rendering of Scratched Metal by Deforming MDF of Normal Mapped Anisotropic Surface

Young-Min Kang
Tongmyong University
ymkang@tu.ac.kr

Hwan-Gue Cho
Pusan National University
hgcho@pusan.ac.kr

Sung-Soo Kim
ETRI
sungsoo@etri.re.kr

ABSTRACT

An effective method to render realistic metallic surface in realtime application is proposed. The proposed method perturbs the normal vectors on the metallic surface to represent small scratches. General approach to the normal vector perturbation is to use bump map or normal map. However, the bumps generated with those methods do not show plausible reflectance when the surface is modeled with a microfacet-based anisotropic BRDF. Because the microfacet-based anisotropic BRDFs are generally employed in order to express metallic surface, the limitation of the simple normal mapping or other normal vector perturbation techniques make it difficult to render realistic metallic object with various scratches. The proposed method employs not only normal perturbation but also deformation of the microfacet distribution function (MDF) that determines the reflectance properties on the surface. The MDF deformation enables more realistic rendering of metallic surface. The proposed method can be easily implemented with GPU programs, and works well in realtime environments.

Keywords: Realtime rendering, anisotropic reflectance, metal rendering, MDF deformation

1 INTRODUCTION

In this paper, we propose a procedural method that efficiently renders plausible metallic surfaces as shown in Fig.1. Anisotropic reflectance models have been widely employed to represent the metallic surface. However, realistic representation of small scratches shown in Fig.1 were not main concern of those methods.

Torrance and Sparrow proposed microfacet-based rendering model where the surface to be rendered was assumed as a collection of very small facets[12]. Each facet has its own orientation and reflects like a mirror. The reflectance property of this surface model is determined by microfacet distribution function(MDF).

Many researchers improved the microfacet-based rendering model to represent various materials. Methods that can control the roughness of the surface were introduced[4, 3], and those methods were also improved by Cook and Torrence[5].

A smooth metallic surface reflects the environments like a mirror. However, the most metal objects have brushed scratches or random scratches. Theses scratches make the reflectance on an actual metallic surface different from that on the perfect mirror surface. The peculiar reflectance on metallic surface is determined by the direction of the scratches, and in most cases, has anisotropic appearance. There have been various techniques for representing the anisotropic reflectance[8, 14, 11].

Figure 1: Realtime rendering with proposed method.

Ashikhmin and Shirley proposed an anisotropic reflection model with intuitive control parameters[1, 2]. Their model is successfully utilized to express the surface with brushed scratches.

Wang *et al*. proposed a method that approximates the measured BRDF(bidirectional reflectance distribution function) with multiple spherical lobes[13]. Although this method is capable of reproduce various materials including metallic surface, it has a serious disadvantage in that expensive measured BRDF is required. Moreover, it is still impossible to accurately render small scratches and light scattering with camera close up to the surface.

Although there have been many approaches to representation of metallic surface [15], relatively little attention has been given to the representation of the small scratches on the surface and the reflectance disturbance caused by the scratches. In most cases, only the reflectance anisotropy caused by the scratches was modeled. An efficient and accurate computation of specular reflection has been also introduced for realtime applications[9]. However, it cannot be applied to normal mapped surface because the method is based on vertex geometry.

In this paper, we propose a procedural method that does not require any measured data. The proposed method efficiently and plausibly renders the small scratches and its light scattering on anisotropic reflectance surfaces.

2 REALISTIC METAL RENDERING

In this section, a procedural approach to metallic surface rendering is proposed. The proposed method is based on microfacet model, and the small scratches on the surface are represented with normal vector perturbation. In order to increase the realism, we also deform the MDF according to the perturbation of the normal vector.

2.1 MDF for Anisotropic Reflectance

The reflectance property of microfacet-based surface model is determined by the microfacet distribution function(MDF) $D(\omega_h)$ which gives the probability that a microfacet is oriented to the direction ω_h. Ashikhmin *et al*. proposed an anisotropic reflectance model with the following MDF:

$$D(\omega_h) = \frac{\sqrt{(e_x+1)(e_y+1)}}{2\pi}(\omega_h \cdot \mathbf{n})^{e_x \cos^2\phi + e_y \sin^2\phi} \quad (1)$$

, where $\mathbf{n}$ is the normal vector at the point to be rendered. The actual parameter ω_h in the MDF is the half way vector between the incident light direction and outgoing viewing direction. e_x and e_y are parameters that control the anisotropy of the reflection, and ϕ is the azimuthal angle. ω_h is a unit vector which is sufficiently represented with only two components as $(\omega_h.x, \omega_h.y, \sqrt{1-\omega_h.x^2-\omega_h.y^2})$. Therefore,the MDF is also defined in 2D space as shown in Fig.2.

Fig.2 shows an example of anisotropic MDF using Eq.1 with different e_x and e_y. As shown in Fig.2, the incoming light energy is scattered differently in x(tangent) and y(binormal) axes of tangent space. Such anisotropic reflectance is appropriate for metal rendering. In this paper, we assume that metallic surfaces reflect light energy according to the anisotropic model described in Eq.1

Fig.3 shows the rendering results by changing the parameters e_x and e_y of Eq.1. As shown in the figure, the anisotropic reflectance on metallic surface can be easily controlled. However, this method is not capable of capturing the small scratches and the light scattering in details when the camera is moved close to the surface. A simple approach to this problem is to perturb the normal vectors on the surface, but the perturbed normal vectors on anisotropic reflection surface may introduce another problem. The limitation of simple normal perturbation is described in the next subsection.

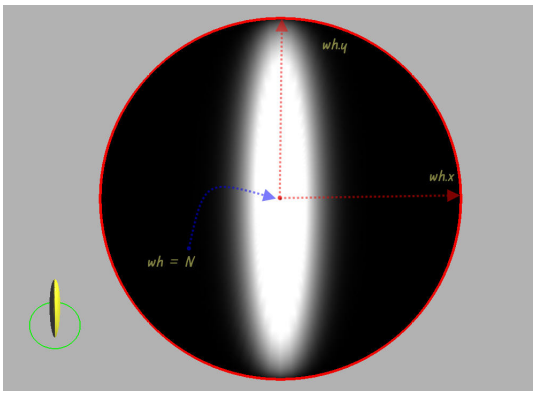
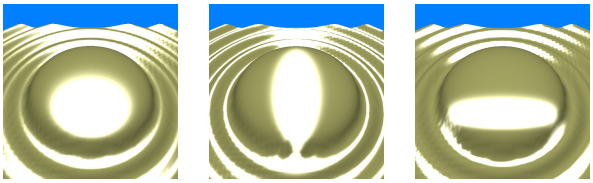


Figure 2: MDF in 2D space



(a) $e_x, e_y : 20, 20$ (b) $e_x, e_y : 200, 10$ (c) $e_x, e_y : 10, 200$

Figure 3: Surfaces rendered with Eq.1: (a) isotropic, (b)&(c) anisotropic reflectance.

2.2 Limitation of Normal Perturbation

There have been continuous efforts to represent higher geometric complexity with simple mesh by perturbing the normal vectors[10, 6, 7]. Bump mapping is well known in graphics literature, normal mapping is an improved method which does not compute normal vectors during the rendering phase[10].

In this paper, we are interested in representing the light scattering by the small scratches on the anisotropic reflection surface. In order to represent the scratches we employed the well-known normal map approach. Fig.4 shows the scratch maps (essentially normal maps), and the expected rendering results. The scratch maps are seamless textures and procedurally generated.

Heidrich and Seidel applied Blinn-Phong shading to the normal mapped geometry[6]. Their method is successful only when the reflection is isotropic. However, the normal mapping on anisotropic reflection surface, unfortunately, cannot reproduce the original anisotropic reflectance on the distorted surface. Other normal perturbation methods such as displacement mapping also suffer from the same problem. Fig.5 shows the un-

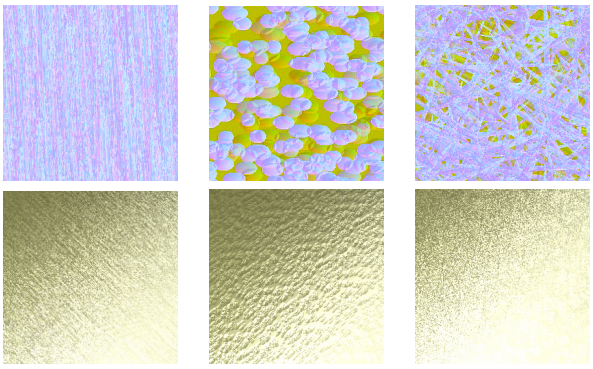
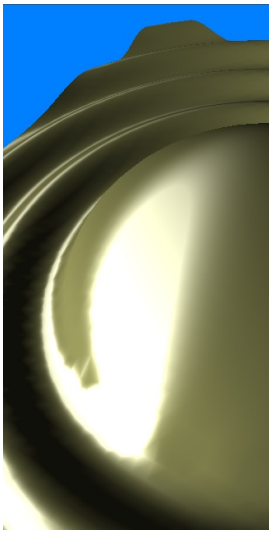
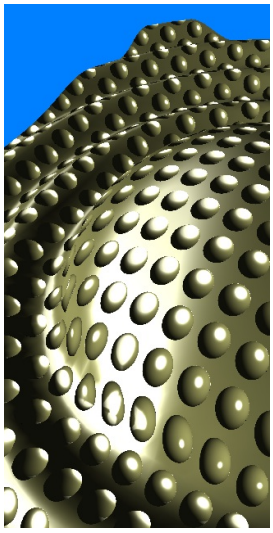


Figure 4: Scratch maps and expected rendering results: (top row) scratch maps and (bottom row) expected results.



(a) original surface



(b) normal mapped surface

Figure 5: Normal vector perturbation on an anisotropic reflection surface: (a) original surface and (b) normal mapped surface.

satisfactory rendering results when the simple normal mapping is applied to an anisotropic reflection surface with MDF function shown in Eq.1. As shown in the figure, the anisotropic reflectance on the original surface (a) is not preserved in the normal mapped surface (b). The reflectance on the area where normal vectors are perturbed is rather isotropic. Moreover we can observe some artifacts that specular reflection is severely distorted at the left lower region.

The problem shown in Fig.5 is because the normal mapping or other normal vector perturbation methods only change the normal vector $\mathbf{n}$. However, the MDF $D(\omega_h)$ is dependent not only on $\mathbf{n}$ but also on ω_h. In Eq.1, the only argument was ω_h because the normal vector is constant in tangent space. However, the normal vector should be another argument when normal perturbation is applied. Let us denote the perturbed normal vector as $\tilde{\mathbf{n}}$. The MDF can then be rewritten as follows:

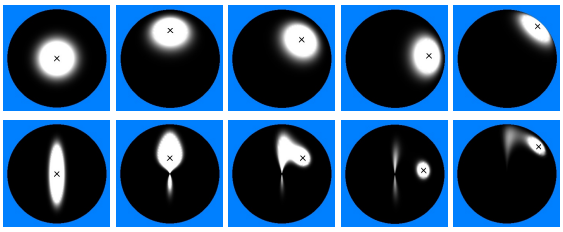


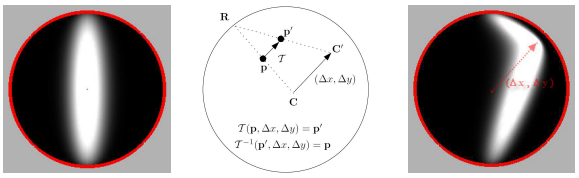
Figure 6: MDF with perturbed normal vectors: (top row) perturbation with isotropic MDF and (bottom row) perturbation with anisotropic MDF.

$$D(\omega_h,\tilde{\mathbf{n}}) = \frac{\sqrt{(e_x+1)(e_y+1)}}{2\pi}(\omega_h\cdot\tilde{\mathbf{n}})^{e_x\cos^2\phi+e_y\sin^2\phi} \quad (2)$$

Heidrich and Seidel computed the dot product of half way vector and the perturbed normal vector to calculate the specular reflection on the normal mapped surface. Eq.2 also computes the dot product. However, this method does not work well for anisotropic reflection surface. Fig.6 shows the MDF computed with Eq.2 and perturbed normal vectors. The cross mark in the figure indicates the perturbed normal. The top row of Fig.6 shows isotropic MDF when the normal vector is perturbed. As shown in the figure, Eq.2 produces reasonable deformed MDF for the isotropic MDF. However, the simple normal perturbation is not successful with anisotropic MDFs. The bottom row of fig.6 shows the results when we employed an anisotropic MDF. The results show that simple normal perturbation approach is hopelessly unsuccessful to preserve the original reflection property.

2.3 MDF Deformation

In order to overcome the limitation of the simple normal mapping on anisotropic reflection surface, the MDF should be properly deformed with the original anisotropic property maintained. Fig.7 shows the MDF deformation concept. Fig.7 (a) shows an example of anisotropic MDF, and (c) shows the deformed MDF in accordance with the normal vector perturbation amount of $(\Delta x,\Delta y)$ in tangent space. Let us denote the deformed MDF as $D'(\omega_h)$. We can easily derive $D'(\omega_h)$ with the deformation concept shown in Fig.7 (b). A certain point $\mathbf{p}$ in the domain of the original MDF $D(\omega_h)$ must move to another location $\mathbf{p}'$ in the domain of the deformed MDF $D'(\omega_h)$. The direction and magnitude of the movement are determined by the movement from the center of the original MDF space ($\mathbf{C}$) to that of the deformed MDF space ($\mathbf{C}'$). The movement of the center is in fact the perturbation of the normal vector, and can be denoted as $(\Delta x,\Delta y)$. Let us denote the transformation that move a point from $\mathbf{p}$ to $\mathbf{p}'$ in accordance with the normal perturbation $(\Delta x,\Delta y)$ as $\mathcal{T}(\mathbf{p},\Delta x,\Delta y)$. The transformation $\mathcal{T}(\mathbf{p},\Delta x,\Delta y)$ can be easily derived with $\mathbf{R}$, the intersection of the



(a) original MDF (b) deformation (c) deformed MDF

Figure 7: MDF deformation concept and corresponding points.

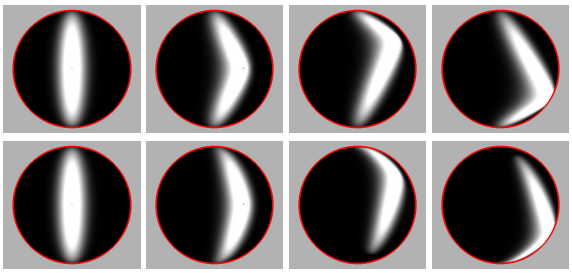


Figure 8: MDF deformation examples: (top row) linear interpolation results and (bottom row) smooth interpolation results.

circumference of the MDF space and the ray from the center through the point **p**.

The simple approach shown in Fig.7 move the point **p** in the same direction with the center movement, and the magnitude of the movement is linearly interpolated. Therefore, the transformation can be expressed as follows:

$$\mathscr{T}(\mathbf{p},\Delta x,\Delta y) = \mathbf{p} + \frac{|\vec{\mathbf{Rp}}|}{|\vec{\mathbf{RC}}|}(\Delta x,\Delta y) \tag{3}$$

Although the transformation shown in Eq.3 deforms the MDF in accordance with the normal vector perturbation, the bending of the deformed anisotropic reflectance is excessive at the moved center as shown in Fig.7 (c). In order to obtain more smooth interpolation, we used the following transformation:

$$\mathscr{T}(\mathbf{p},\Delta x,\Delta y) = \mathbf{p} + \sqrt{\frac{|\vec{\mathbf{Rp}}|}{|\vec{\mathbf{RC}}|}}(\Delta x,\Delta y) \tag{4}$$

Fig.8 compares the MDF deformation results with the linear (Eq.3) and the smooth (Eq.4) interpolations. The top row shows the linear version while the bottom row shows the smooth version. As shown in the figure, the smooth interpolation version looks more natural.

It is obvious that computing the deformed MDF at each sampling point on the surface is extremely inefficient. Explicit deformation of the MDF is only conceptual process. In the actual rendering process, we never compute $D'(\omega_h)$. Only the original MDF $D(\omega_h)$ is used with the inverse transformation $\mathscr{T}^{-1}(\mathbf{p}',\Delta x,\Delta y)$. In other words, we conceptually employ $D'(\omega_h)$ for the normal mapped surface, but actually use $D(\mathscr{T}^{-1}(\omega_h,\Delta x,\Delta y))$ which has the equivalent value.

The inverse transformation of Eq.4 can be easily obtained as follows:

$$\mathscr{T}^{-1}(\mathbf{p}',\Delta x,\Delta y) = \mathbf{p}' - \sqrt{\frac{|\vec{\mathbf{Rp}}'|}{|\vec{\mathbf{RC}}'|}}(\Delta x,\Delta y) \tag{5}$$

Now we can simply calculate $D(\mathscr{T}^{-1}(\omega_h,\Delta x,\Delta y))$ to compute the MDF at the point where the normal vector is perturbed with $(\Delta x,\Delta y)$. Because Δx and Δy are the x and y components of the perturbed normal vector, $D(\mathscr{T}^{-1}(\omega_h,\Delta x,\Delta y))$ can be also rewritten as $D(\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}))$.

It should be noted that the MDF with the inverse transformation, i.e., $D(\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}))$, still remain in the original MDF space. The normal vector is always (0,0,1) in tangent space. Therefore, the dot product of any vector **v** and the normal vector **n** (i.e., $\mathbf{v}\cdot\mathbf{n}$) is simply the z component of the vector, $\mathbf{v}.z$, and the actual MDF we used is as follows:

$$\begin{aligned} D'(\omega_h,\tilde{\mathbf{n}}) = & \\ D(\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}),\mathbf{n}) = & \quad \frac{\sqrt{(e_x+1)(e_y+1)}}{2\pi}\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}).z^{\varepsilon} \end{aligned} \tag{6}$$

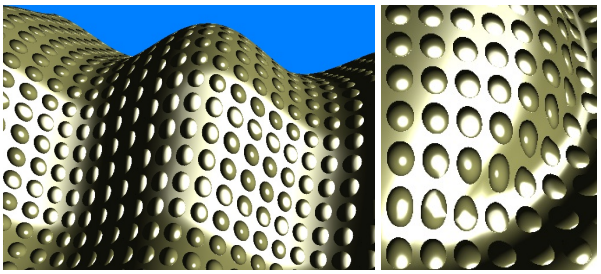
,where the exponent ε is $e_x\cos^2\phi + e_y\sin^2\phi$.

Fig.9 shows the effect of the MDF deformation by comparing the specular reflections on the illusory bumps. The bumpy illusion on the surface shown in Fig.9 (a) is generated only with normal mapping method while the result shown in Fig.9 (b) is generated with MDF deformation techniques. The original surface has anisotropic reflection property. However, as shown in the figure, the original MDF does not reproduce the anisotropic reflectance on the bumps. Even worse, the shapes of the specular reflection areas are weirdly distorted on some bumps. The deformed MDF removes such disadvantages as shown in Fig.9 (b). The anisotropic reflectance is well preserved on each illusory bump, and no weird shapes are found.

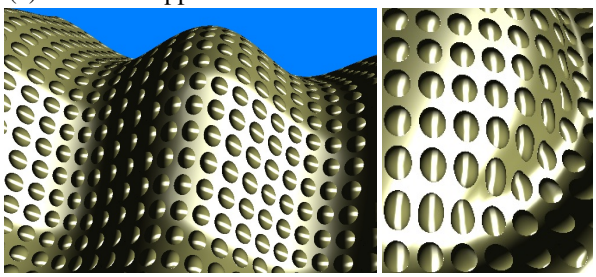
2.4 Scratch Map Generation

As mentioned earlier, we represent the natural metallic appearance by engraving small scratches on the surface. Those scratches are expressed with perturbed normal vectors, and some example normal maps were already shown in Fig.4.

The scratch maps can be generated with various techniques, but it can be easily and efficiently created in a procedural manner. In order to devise a scratch map generation method, we employed engraving a hemisphere as a basic operation. The normal vectors on the engraved hemispherical surface can be easily computed

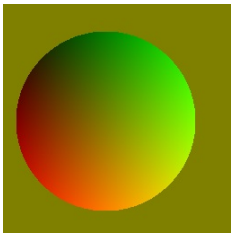


(a) Normal mapped surface without MDF deformation

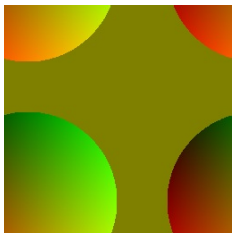


(b) Normal mapped surface with deformed MDF

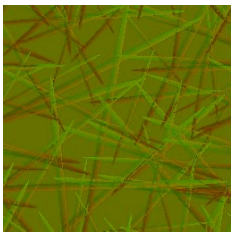
Figure 9: Effect of MDF deformation on anisotropic reflection surface: normal mapping (a) without MDF deformation and (b) with additional MDF deformation applied.



(a) basic pit



(b) moved pit

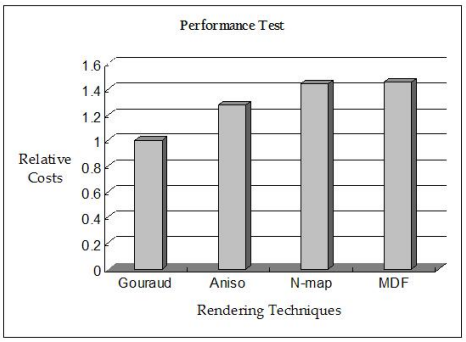


(c) random direction

(d) directional tendency

Figure 10: Concept of scratch map generation

in tangent space. Fig.10 (a) shows the basic scratch texture with one engraved hemisphere. The center of the hemisphere can freely move within the texture space. We made our texture seamless as shown in Fig.10 (b). We can also scale the hemisphere and stretch in any direction, and arbitrarily increase the number of engraved pits. The depth of the engraved scratch can be also arbitrarily changed. Fig.10 (c) and (d) show the scratch maps generated by stretching the engraved pits in random direction and in a certain range of directions respectively.



Tech	Gouraud	Aniso	N-Map	MDF
Cost	1	1.28	1.44	1.46

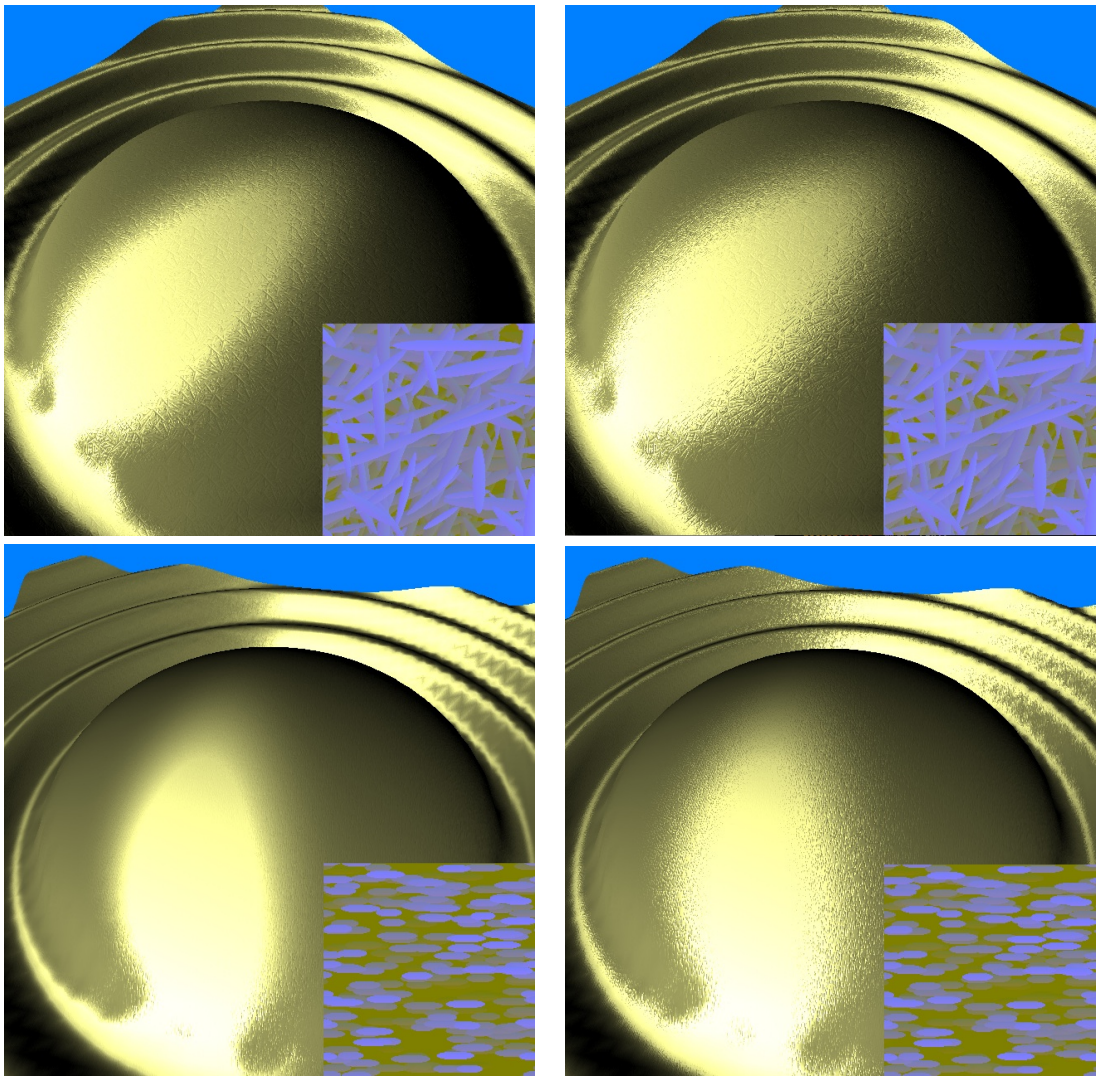
Figure 11: Rendering performance of the proposed method compared with other realtime methods.

3 EXPERIMENTS

The techniques proposed in this paper was implemented with OpenGL shading language, and the computing environments were Mac OS X operating system with 2.26 GHz Intel core 2 CPU, 2 G DDR3 RAM and NVIDIA 256M VRAM GeForce 9400M. Fig.11 is the performance analysis of the proposed method compared with previous traditional approaches. The label 'Aniso' means Ashikhmin-Shirley anisotropic reflection model, 'N-map' represents normal mapping, and 'MDF' indicates the proposed MDF deformation techniques. The computational cost of Gouraud shading is taken as a unit cost, and other rendering techniques were compared with the unit cost. As shown in the figure, the proposed method with deformed MDF is just slightly more expensive than usual normal mapping (labeled as N-Map in the figure) which works very well in realtime environments.

Fig.12 compares the light scattering on normal mapped anisotropic reflection surface. Fig.12 (a) shows the rendering results where normal mapping is applied without deforming the MDF while (b) shows results rendered with additional MDF deformation. The normal map image in the right bottom corner is the scratch map applied. As shown in the figure, the scratches represented by simple normal mapping do not plausibly scatter the light. However, the results with the proposed method in (b) show realistic light scattering along the rim of the specular reflection area.

Fig.13 shows the effect of the MDF deformation when environments are mapped on the surface. The reflection on the surface is modeled with Ashikhmin and Shirley BRDF model. The left column of the Fig.13 shows the result without the environment mapping while the right column shows the rendering results with environment mapping. The first row in the figure shows the original anisotropic reflection surface of Ashikhmin and Shirley's model with the scratch map texture in the right bottom corner. The middle row



(a) normal mapping (b) MDF deformation

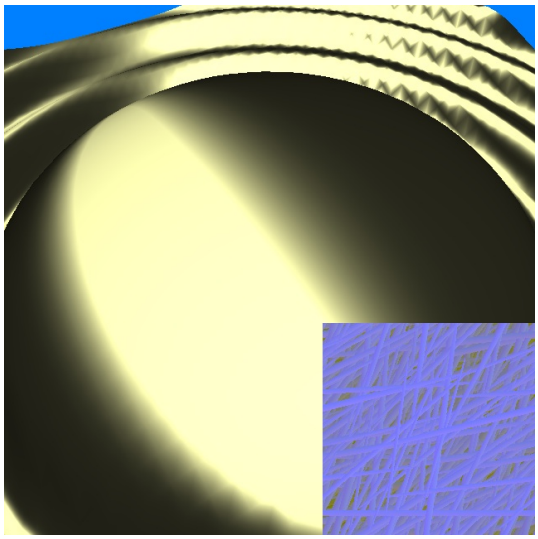
Figure 12: Comparison of light scattering on (a) simple normal mapped surface and (b) normal mapped surface with additional MDF deformation.

shows the results only with the simple normal mapping, and the bottom row shows the result when the proposed MDF deformation is additionally applied. As shown in the figure, the additional MDF deformation increases the rendering quality, and reproduces the light scattering by the scratches.

Although, in this paper, we employed Ashikhmin and Shirley BRDF for modeling the anisotropic reflection surface, the proposed method works with any anisotropic reflection surface. For example, our method works better with Ward BRDF model. The Ward BRDF is also an anisotropic reflection model[14].

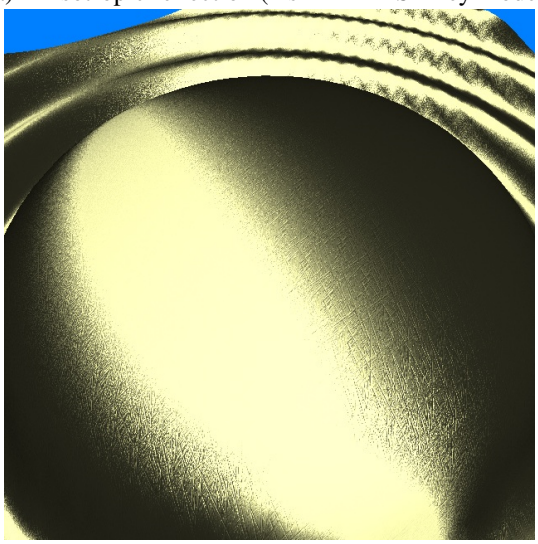
Fig.14 shows the effect of the proposed method when the surface is model with Ward anisotropic BRDF. The reflection on the surface is modeled with Ward anisotropic BRDF model. The left column of the Fig.14 shows the result without the environments mapping while the right column shows the rendering results with environments mapping. The first row in the figure shows the original anisotropic reflection surface of Ward BRDF model. The middle row shows the results only with the simple normal mapping, and the bottom row shows the results when the proposed MDF deformation is additionally applied. As shown in the figure, the simple normal mapping on Ward BRDF surface does not provide plausible light scattering. In fact, the effect of the perturbed normal vector can be hardly observed without environment mapping. Only when the proposed method is applied, we can obtain plausible light scattering on the scratched surface as shown in the bottom row.

Fig.15 shows the close-up comparison of light scattering effects of simple normal mapping and the proposed method. The results shown in (a) and (b) were rendered with Ward BRDF for anisotropic reflection on the surface while Ashikhmin and Shirley BRDF model

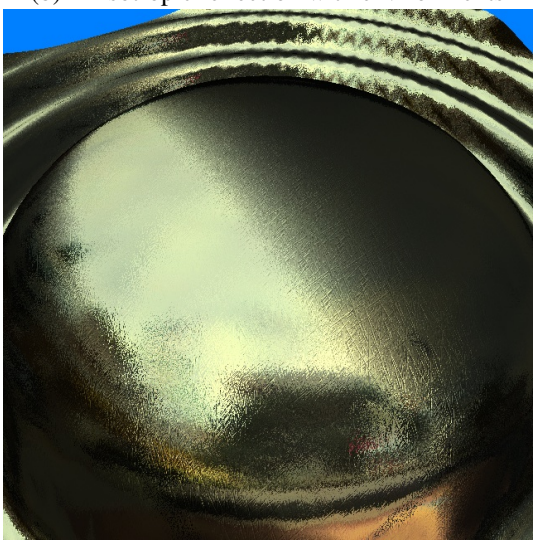


(a) Anisotropic reflection (Ashikhmin-Shirley model)

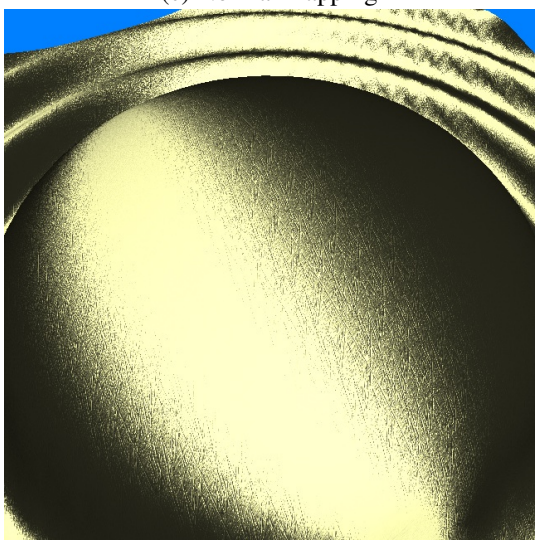
(b) Anisotropic reflection with environments



(c) Normal mapping



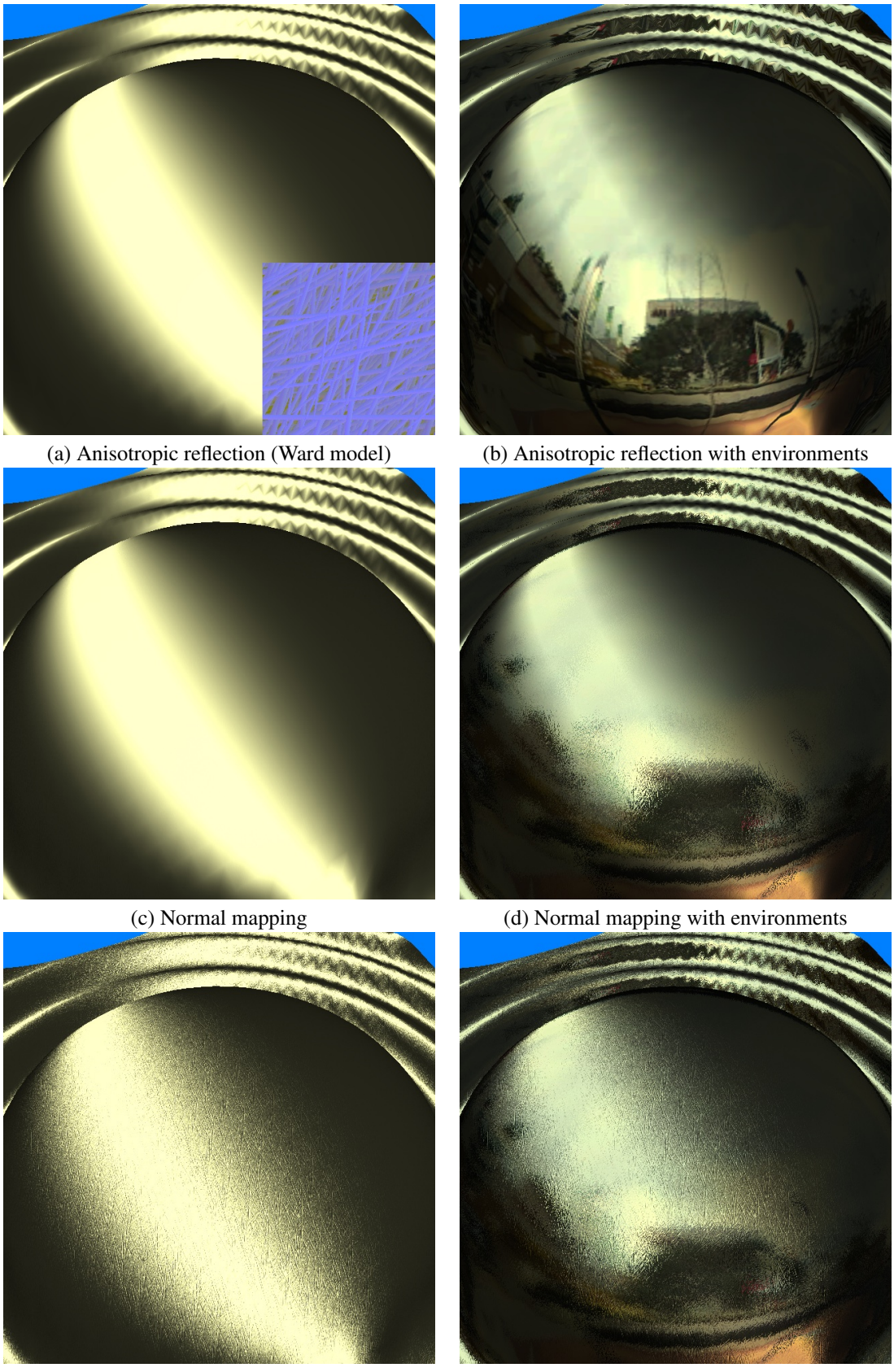
(d) Normal mapping with environments



(e) MDF deformation

(f) MDF deformation with environments

Figure 13: The effect of the propose method on Ashikhmin and Shirley model: (left column) no environment mapping, (right column) environment mapping, (top row) original anisotropic reflection surface, (b) normal mapping, and (c) normal mapping with MDF deformation.



(a) Anisotropic reflection (Ward model)

(b) Anisotropic reflection with environments

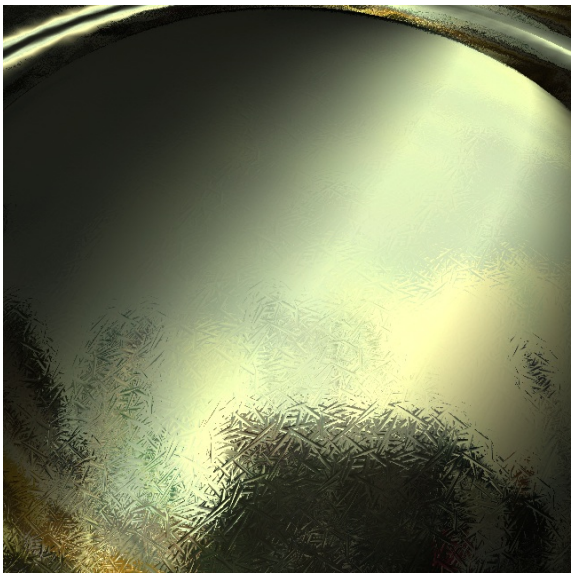
(c) Normal mapping

(d) Normal mapping with environments

(e) MDF deformation

(f) MDF deformation with environments

Figure 14: The effect of the propose method on Ward's model: (left column) no environment mapping, (right column) environment mapping, (top row) original anisotropic reflection surface, (b) normal mapping, and (c) normal mapping with MDF deformation.



(a) Normal mapping on Ward BRDF surface

(b) MDF deformation on the Ward surface

(c) Normal mapping on Ashikhmin-Shirley BRDF surface

(d) MDF deformation on the Ashikhmin-Shirley surface

Figure 15: Close-up comparison of light scattering: (a) simple normal mapping on a surface with Ward anisotropic reflection model, (b) additional MDF deformation applied on the Ward model, (c) simple normal mapping on Ashikhmin-Shirley BRDF surface, and (d) MDF deformation effect on the Ashikhmin-Shirley surface.

is employed for those shown in (c) and (d). Fig.15 (a) and (c) show the results only with the normal mapping while (b) and (d) are results generated with the proposed MDF deformation method. As shown in the figure, normal mapping with deformed MDF shows superior rendering quality to the simple normal mapping approach.

4 CONCLUSION

In this paper, we proposed an effective and efficient method that improves the normal mapping to be successfully applied to anisotropic reflection surfaces. The proposed method is appropriate for rendering metallic surfaces with small scratches in realtime. We have shown in this paper that the simple normal mapping or other normal perturbation techniques cannot be applied to anisotropic reflection surfaces. In order to enable normal perturbation to better illusory bumps on surface, we introduced MDF deformation concept. The experimental results show that the proposed method achieves far better rendering quality than simple normal mapping method does. Moreover, the computational cost additionally required for MDF deformation is small enough for realtime environments. The only difference between the proposed method and the traditional anisotropic BRDF models is that ω_h given to the MDF is adjusted. Therefore, the proposed method is easily implemented as GPU program and works well in

realtime environments. The proposed method can be successfully utilized in games or virtual reality systems for rendering high-quality metallic surfaces.

ACKNOWLEDGEMENTS

This work was supported in part by the SW computing R&D program of MKE/KEIT [10035184], "Game Service Technology Based on Realtime Streaming".

REFERENCES

[1] M. Ashikhmin, S. Premoze, and P. Shirley. A microfacet-based brdf generator. *In Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 65–74, 2000.

[2] M. Ashikhmin and P. Shirley. An anisotropic phong brdf model. *Journal of Graphics Tools*, 5(2):25–32, 2002.

[3] J. Blinn. Models of light reflection for computer synthesized pictures. *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, 1977.

[4] J. Blinn and M. Newell. Texture and reflection in computer generated images. *Communication of ACM*, 19(10):542–547, 1976.

[5] R. L. Cook and K. E Torrance. A reflectance model for computer graphics. *Computer Graphics (ACM Siggraph '81 Conference Proceedings)*, 15(3):307–316, 1981.

[6] W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. *In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–178, 1999.

[7] M. Pharr and G. Humphreys. In *Physically-based Rendering*. Elsevier (Morgan Kaufman Publishers), 2004.

[8] M. Poulin and A. Founier. A model for anisotropic reflection. *Computer Graphics (ACM Siggraph '90 Conference Proceedings)*, 23(4):273–282, 1990.

[9] D Roger and N. Holzschuh. Accurate specular reflections in real-time. *Computer Graphics Forum*, 25(3):293–302, 2006.

[10] H. Rushmeier, G. Taubin, and A. Gueziec. Applying shapes from lighting variation to bump map capture. *In Proceedings of Eurographics Rendering Workshop '97*, pages 35–44, 1997.

[11] C. Schilick. A customizable reflectance model for everyday rendering. *In Proceedings of the 4th Eurographics Workshop on Rendering*, pages 73–84, 1993.

[12] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 57(9), 1967.

[13] J. Wang, P. Ren, M. Gong, J. Snyder, and B. Guo. All-frequency rendering of dynamic, spatially-varying reflectance. *In Proceedings of ACM Siggraph Asia 2009*, pages 1–10, 2009.

[14] G. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics (ACM Siggraph '92 Conference Proceedings)*, 26(2):265–272, 1992.

[15] L zirmay Kalos, T. Umenhoffer, Gustavo Patow, L. Szecsi, and M Sbert. Specular effects on the gpu: State of the art. *Computer Graphics Forum*, 28(6):1586–1617, 2009.

Multiscale Visualization of 3D Geovirtual Environments Using View-Dependent Multi-Perspective Views

Sebastian Pasewaldt Matthias Trapp Jürgen Döllner
Hasso-Plattner-Institut, University of Potsdam, Germany
{sebastian.pasewaldt|matthias.trapp|juergen.doellner}@hpi.uni-potsdam.de

ABSTRACT

3D geovirtual environments (GeoVEs), such as virtual 3D city models or landscape models, are essential visualization tools for effectively communicating complex spatial information. In this paper, we discuss how these environments can be visualized using multi-perspective projections [10, 13] based on view-dependent global deformations. Multi-perspective projections enable 3D visualization similar to panoramic maps, increasing overview and information density in depictions of 3D GeoVEs. To make multi-perspective views an effective medium, they must adjust to the orientation of the virtual camera controlled by the user and constrained by the environment. Thus, changing multi-perspective camera configurations typically require the user to manually adapt the global deformation — an error prone, non-intuitive, and often time-consuming task. Our main contribution comprises a concept for the automatic and view-dependent interpolation of different global deformation preset configurations (Fig. 1). Applications and systems that implement such view-dependent global deformations, allow users to smoothly and steadily interact with and navigate through multi-perspective 3D GeoVEs.

Keywords: multi-perspective views, view-dependence, global space deformation, realtime rendering, virtual 3D environments, geovisualization.

1 INTRODUCTION

3D GeoVEs, such as virtual 3D city and landscape models, represent efficient tools for fields such as geography or cartography, in particular if their visualization and knowledge can be transferred to the 3D visualization domain [9]. Previous work has shown that global deformation applied to such environments can be used to assist wayfinding and navigation by making effective use of the available image space [10, 13] and by reducing occlusions [18]. Grabler et al. [2009] demonstrate that the usage of multi-perspective views in combination with cartographic generalization techniques such as simplification and deformation is suitable to convey important information with in 3D tourist maps.

In the context of interactive global deformations and multi-perspective views, existing visualization techniques and systems are most effective for specific settings of a virtual camera, i.e., Fig. 2. The virtual camera must be near the ground (pedestrian view) or at a certain height (birds-eye view), in order to exploit the full potential of these visualization techniques. Usually, in a 3D GeoVE the user wants to interact and navigate freely. This would require the manual adaptation of the visualization parameters during interaction and navigation. In general, this task is complex, error-prone, and time-consuming. In this paper we develop a concept that delivers a suitable visualization for a camera setting via automatic view-dependent interpolation of global deformations that are represented by parametric curves.

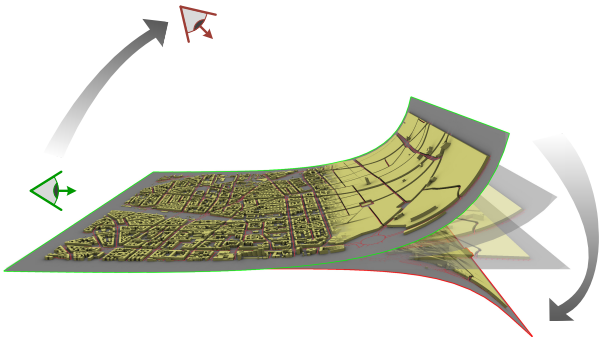


Figure 1: Conceptional sketch of the interpolation of global deformations and different geometric representations based on the viewing angle of the virtual camera.

Further, a drawback of 3D GeoVEs are the multiple geometric scales [9], introduced by the perspective projection of the camera, because they lead to small scales in the more distant parts of the scene. Consequently, the depiction of objects only have limited image space (e.g. only one pixel) and cannot be distinguished by a viewer (pixel noise). To overcome this problem in the domain of paper maps, cartographers apply generalization techniques to minimize visual complexity and to improve comprehension. A similar concept is used in most of the current multi-perspective techniques. Instead of using a photo-realistic style, a map-based style is applied to regions of small scales. We generalize the style concept

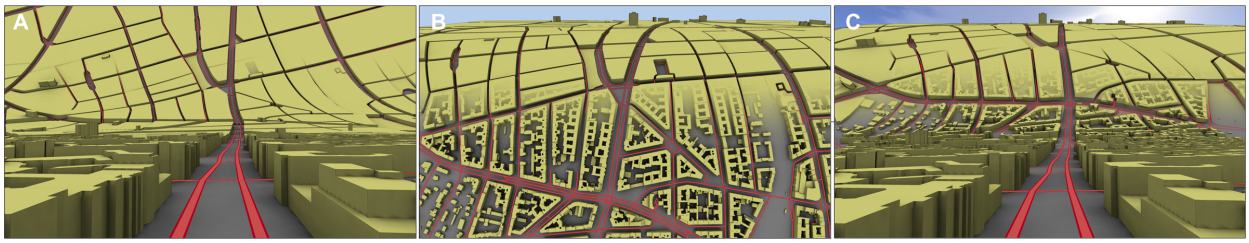


Figure 2: Exemplary results of our visualization system that enables the view-dependent interpolation of the depicted scenes: progressive perspective (A), degressive perspective (B), and a hybrid perspective (C) using different generalization levels of a 3D virtual city model of Berlin.

by letting the user define multiple geometric representations, e.g., obtained from cell-based generalization [8], to sections of the curve (Fig. 2). Further, these explicit geometric representations enable more design freedom then automatically derived style variations such as in [10]. Jobst and Döllner (2008) further suggest to subdivide the visualization into zones where a constant scaling and thus a constant generalization is applied per zone. An exemplary visualization can be seen in Fig. 7.

Möser et al. [13] generalize the concept introduced in [10] by using Hermite curves for the parameterization of global deformations, which can be easily manipulated by the user. However, the application of standard parameterized curves for such a visualization introduces additional geometric distortions. We compensate these by an arc-length parameterization [14].

In this work we present a concept and system that addresses the above challenges with respect to realtime raster-based graphics synthesis. To summarize, this work makes the following contribution:

1. It describes a concept for the automated and view-dependent interpolation of global deformations based on the viewing angle of the user's virtual camera with respect to a reference plane.

2. It further presents an extension to global deformations that enables a user to define different geometric representations for different sections along a deformation curve and enables their image-based interpolation.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 introduces the concept of view-dependent global deformations. Section 4 describes steps to prepare the visualization. Section 5 outlines how to implement the concept as a realtime rendering technique. Section 6 consists of a performance evaluation, a preliminary user study, discusses problems and limitations, and presents ideas for future work. Section 7 concludes this paper.

2 RELATED WORK

Panoramic Imaging

Panoramic maps were introduced by H.C. Berann [3]. He combined handcrafted geographic with terrestrial depictions and different projection techniques to generate a new kind of map, which assists the user in the orientation task. This work was time-consuming and tedious. Premoze introduced a framework for the computer aided generation of panoramic maps [17]. It offers tools to assist the map-maker in the work flow of the hand-tailored maps. A semi-automatic approach to generate panoramic maps, which relies on global deformations, is presented in [24]. Falk et al. introduced a semi-automatic technique based on a force field that is extracted from the terrain surface [7]. Degener & Klein concentrate on parameters like occlusion and feature visibility in their automatic generation of panoramic maps [6]. All approaches combine non-linear perspectives in one final image, but rely on different techniques.

Non-linear Perspectives

Non-linear Perspectives can be achieved with different techniques: (1) Using non-standard, non-linear projection to produce a non-linear perspective image, or combine several images taken from different perspectives ([1], [23]). (2) Reflection on non planar surfaces and (3) Local or global space deformation [26]. The combination of different images to one final image as used in [1] and [23] can also be expressed by a space-deformation as introduced in [2]. The Single Camera Flexible Projection Framework of [4] is capable of combining linear, non-linear and handmade projections in realtime. The projections are described by a deformed viewing volume. Similar to free-form deformation (FFD [22]), the view frustum serves as lattice. Objects or viewing rays are deformed according to the deformation of the lattice. For the occlusion free visualization of driving routes Takahashi et al. rely on global space deformation [25].

On the one hand the mentioned techniques offer a broad and flexible definition of the projections, which enables the user to control nearly every facet of the final perspective. On the other hand a large number of non-intuitive parameters have to be controlled. Brosz et al. [2007] abstracts from these parameter by using a lattice. Similarly, we rely on a 2D B-Spline curve to control the 3D curve-based deformation.

Global Deformations

The work of Lorenz et al. [10] uses global deformation to generate non-linear perspectives. The geometry is mapped on two different planes, which are connected by a Bézier surface. The planes may vary in tilt, allowing for a combination of two different perspectives. Similar to panoramic maps, a mixture of cartographic maps and aerial images is used. The different stylization are seamlessly blended in the transition between the planes. Möser et al. [2008] extend this idea by using a more flexible Hermite curve to control the deformation. They also rely on a combination of aerial and cartographic images to apply a kind of generalization in the more distant parts of the scene.

Our approach is based on parametric curves, too. Instead of using a Hermite curve, we decided to use a B-Spline curve, because it offers more flexibility without the need of combining several curves. Furthermore, an arbitrary number of stylizations can be defined, which are not restricted to textures. Instead, we exploit the possibility of blending between different geometric representations generated by the generalization of 3D virtual city models as introduced in [8]. We introduce a view-dependent variation based on the work of Rademacher [19]. He defines key-deformation with associated key viewing points. Depending on the current viewpoint the key-deformations are interpolated. A similar approach is used by [5] for interactive stylized camera control. Another view-dependent variation of deformations is discussed in [12]. Here the global deformation is modified by a view or distant-dependent control function that can depend on a virtual camera.

3 VIEW-DEPENDENT GLOBAL DEFORMATIONS

Our approach consists of two main phases: (1) Rigging Visualization Presets: The user prepares discrete presets of the visualization. One visualization preset includes a deformation curve, the assignment of geometric representations to curve sections (tagging), and the definition of a viewing angle for which the preset is valid. (2) Realtime Visualization: During runtime the presets are interpolated using the camera parameters, which are manipulated during navigation or interaction with the 3D GeoVE.

3.1 Preliminaries

For our visualization we assume that a 3D GeoVE can be approximated by a 3D reference plane $R = (N, O) \in \mathbb{R}^3 \times \mathbb{R}^3$ defined by a normal vector N and a position vector O. Thus, and because of the isotropy of the global deformation variants used in this paper, a view setting for a virtual camera can be described by a viewing angle $\phi = cos(90 - C_D \cdot N)$ (Fig. 4).

To implement progressive or degressive perspectives [10] or hybrid forms [13], our approach uses B-Splines curves [20] instead of Hermite curves. In our experiments we use cubic B-Splines curves ($k = 4$) with four or six control points. In [9] it is argued that a smaller transition zone and linear segments would benefit the comprehension of such a visualization. This specific configuration is hard to implement using a single Hermite curve, but can be easily achieved using B-Splines curves with six control points, by setting two consecutive control points to the same position (Fig. 7).

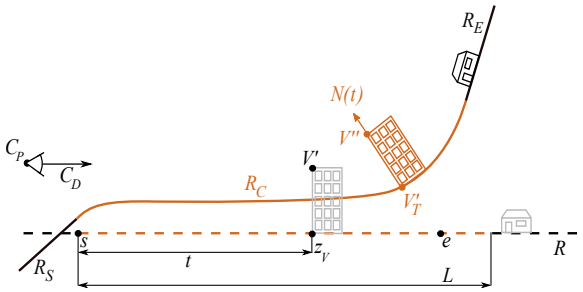


Figure 3: The reference plane R is separated by the parameter s and e into three sections: R_S, R_S and R_C. Based on the depth $z_{V'}$ of the vertex V along the camera direction C_D, the vertex is deformed onto one of the sections.

3.2 Application of Deformation Curves

We apply a global space deformation based on parametric curves, where the curve defines the deformation behavior. Therefore, R is subdivided into three sections (Fig. 3): (1) the curve-controlled section R_C, (2) a planar extension at the start R_S, and (3) a second planar extension at the end R_E. The deformation of R_C is controlled by a B-Spline curve $C(t)$ with a static open knot vector. Assuming that the control points B_i are fixed for a specific B-Spline, the position vector in curve-space $C(t) \in [0,1] \times [1,-1]$ only depends on the parameter t. To deform an input vertex $V = (x, y, z, w) \in \mathbb{R}^4$ we need to establish a mapping between V and t.

To establish the mapping, we first aligned V along the z-axis of the camera space $V' = V \cdot \mathbf{R}_A$. $\mathbf{R}_A$ rotates V around O by ϕ. After the rotation, every vertex is aligned along the viewing direction C_D of the virtual camera. The depth of V' is linearized between the user defined scalars for the start s and end e of the curve in camera space to compute $t \in [0,1]$. To account to the varying arc length L of the B-Spline curve in curve space, we perform a second normalization of t by L (Fig. 3). The rotation during the mapping is necessary, since otherwise a change of ϕ would lead to a different depth value of V and thus to a different mapping between V and t. Finally, the deformed vertex V'' is computed as follows:

$$V'' = \begin{cases} V' \cdot \mathbf{M}_S & t < 0 \\ V' \cdot \mathbf{M}_E & t > L \\ V' \cdot \mathbf{M}_{C(t)} & otherwise \end{cases} \qquad t = \frac{z_{V'} - s}{e - s} \cdot \frac{1}{L}$$

The deformation matrix $\mathbf{M}_{C(t)}$ consists of two separate translations $\mathbf{T}_{C(t)}$ and $\mathbf{D}_{C(t)}$, which are applied to V' se-

quentially. $\mathbf{T}_{C(t)}$ translates the vertex according to its position on the curve: Based on t a position vector $C(t)$ in curve space is computed. $C(t)$ is mapped back to camera space and used to translate V' onto R_C, yielding V'_T. Afterwards $\mathbf{D}_{C(t)}$ translates the vertex along the normal of the curve as follows: Based on the bi-normal B_x and the tangent $C'(t)$ the normal $N(t) = C'(t) \times B_x$ is computed. V'_T is translated along $N(t)$ by a distance d. Here, d denotes the distance of V' to its projection onto R. We just translate the position of the input vertex, because our deformation is a space deformation only. Operations which depends on vertex attributes, e.g. normals, are applied to the undeformed scene.

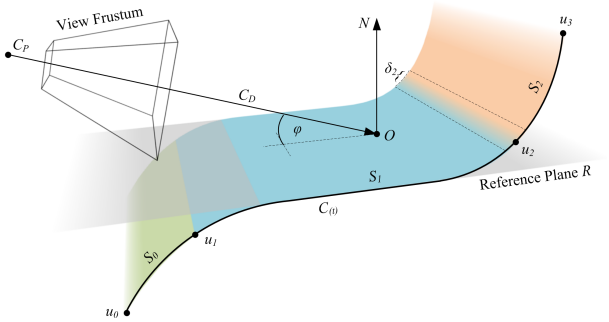


Figure 4: Exemplary parameterization of a deformation curve preset using four tag points (u_i).

To handle the cases of $t \notin [0,1]$ the deformation matrices $\mathbf{M}_S$ and $\mathbf{M}_E$ are applied accordingly to transform V' on R_S or R_E: If the extension plane is parallel to R the matrix is a translation matrix. Otherwise the matrix rotates V' on R_S or R_E. R_S is defined by the normal and position vector of the last B-Spline pont ($C(1)$) and R_E by the first point ($C(0)$).

Depending on the distribution of the control vertices and the knot vector of a B-Spline curve, a sampling with equidistant values t_1, t_2 and t_3 may not yield an equidistant distribution of points $P(t_1)$, $P(t_2)$ and $P(t_3)$, because a B-Spline curve is not arc-length preserving. This is distracting, since it will lead to a scaling error introduced by a straining or stretching of the geometric representation.

To guarantee a correct deformation behavior the curves must be re-parameterized. The approaches of [21] and [15] are not suited for our purposes because they either globally distribute the scaling error or are computational expensive. Instead, we decided to re-parameterize the parameter t similar to the method described in [14]. We sample the B-Spline curve in equidistant intervals and compute the arc-length of these segments. Based on the sampled length L and the according parameter t, the arc-length preserving parameter t' is computed by linear interpolation and stored in a lookup table.

3.3 Visualization Presets

Before we describe the tagging and interpolation of deformation curves, it is necessary to introduce the conceptual term *visualization preset*. As a preset we consider a single perspective (e.g., degressive or progressive). A preset P consists of the following components:

$$P = (C(t), \mathscr{T}, \mathscr{G}, \phi, \tau, s, e, a, b)$$

The set of all presets is denoted as $\mathscr{P}$, with $|\mathscr{P}| = m$. Besides a B-Spline curve $C(t)$ that is used to modify the global deformation, it contains an ordered list of tag points $\mathscr{T}$, a list of geometric representations $\mathscr{G}$ and the following scalar parameters (Fig. 4):

- ϕ: a camera angle, defined through the virtual camera and the reference plane R.
- τ: an angle interval around ϕ, where a preset is valid, i.e., no interpolation of the preset will occur.
- s, e: start and end of the deformation in eye-space. The interval is used to widen or narrow the curve-spaced deformation in camera-direction.
- $a, b \in [0,1]$: start and end of the geometry interpolation. This enables the user to define the geometry interpolation independent from the interpolation of the multi-perspective view.

3.4 Tagging of Deformation Curves

Our system enables the user to associate curve sections with different geometric representations. This can be useful for increasing or decreasing the visual complexity with respect to parts of the visualization. In [10], this was implied by blending between different type of textures within the transition zone and by omitting unimportant buildings. We extend this idea by blending between 3D geometry assigned to consecutive sections of a deformation curve (see Section 5.2). In our examples (Fig. 2 and 7) we use different levels of abstraction (LoA) automatically derived from the virtual city model of Berlin [8].

We can partition a deformation curve $C(t)$ into a number $l \geq 2$ of consecutive *styling sections* as part of the global set of sections $\mathscr{S}$:

$$S_i = (T_i, T_{i+1}, G), \qquad S_i \in \mathscr{S} \qquad G \in \mathscr{G}$$

Here, $i = 0, \ldots, l-1$ represents an index into the list of *tag-points* $\mathscr{T} = T_0, \ldots, T_l$ assigned to every preset P. The geometric representation for a section is denoted as G. A tag point T_i is further defined as follows:

$$T_i = (u, \delta) \quad u, \delta \in [0,1], \quad i = 0, \ldots, l \quad T_i \in \mathscr{T}$$

The position of the tag point on the curve is controlled via the parameter u. δ describes the length of the transition zone between two consecutive sections and is used for blending (see Section 5.2). We assume implicit fixed start and end tag points $T_0 = (0,0)$ at the curves start and $T_l = (1,0)$ at the curves end. Fig. 5 shows the different variants of a terrain model of the grand canyon and the associated active curve preset (inset).

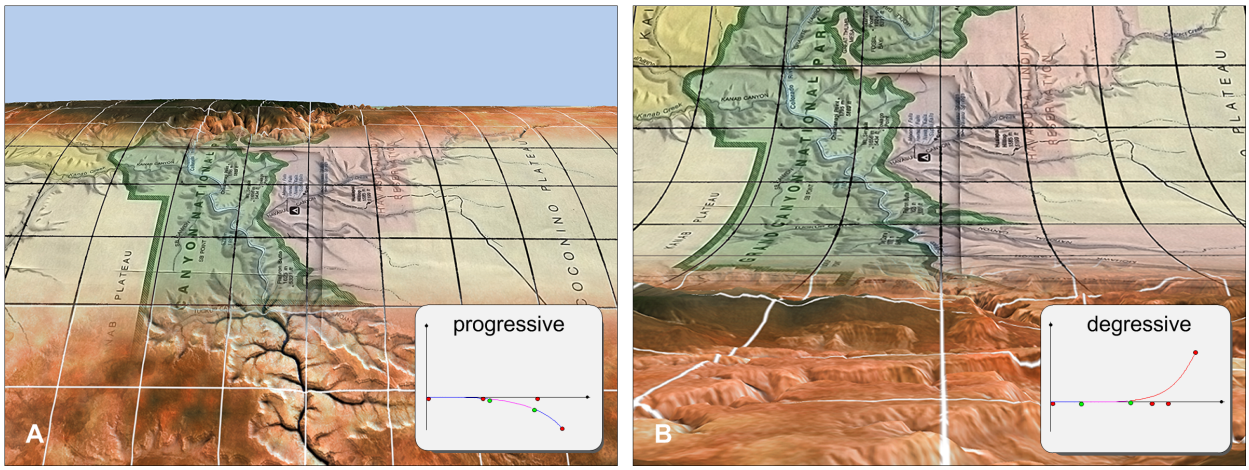


Figure 5: Styling section of a deformation curve with different models of the grand canyon. The inset shows the associated tag point and sections of the curve: The control points are depicted in red and the tag points are depicted green. The grid overlay was added to illustrate the deformation.

3.5 View-Dependent Curve Interpolation

The view-dependent curve interpolation, based on the camera angle ϕ, consists of two main steps: the *preset selection* and the *preset interpolation*. Given the viewing angle of the current virtual camera ϕ_a and the set of all presets $\mathscr{P}$, a selection function $s(\mathscr{P},\phi_a) = (P_S, P_T)$ delivers two presets as follows:

$$s(\mathscr{P},\phi_a) = (P_S,P_T) = \begin{cases} (P_i,P_{i+1}) & \phi_a \geq \phi_i \wedge \phi_a < \phi_{i+1} \\ (P_1,P_2) & \phi_a \leq \phi_1 \\ (P_{m-1},P_m) & \phi_a > \phi_m \end{cases}$$

for all $i = 1,\ldots,m$. This requires an ascending ordering of $\mathscr{P}$ by ϕ performed at the end of the rigging process. Given the viewing angle ϕ_a of the virtual camera and two presets P_S and P_T, the weighting factor σ is calculated as follows:

$$\sigma = clamp\left(\frac{\phi_a - \phi_S}{\phi_T - \phi_S}, 0, 1\right)$$

Given $\sigma \in [0,1]$, the source P_S and target preset P_T, the interpolation $P_I = p(P_S,P_T,\sigma)$ of the current preset P_I is performed by a linear interpolation of all control points: $B_{i,I} = B_{i,P_S} + \sigma \cdot (B_{i,P_T} - B_{i,P_S})$ as well as the respective tag points: $T_{i,I} = T_{i,P_S} + \sigma \cdot (T_{i,P_T} - T_{i,P_S})$.

Beside interpolating the curve related parameters, the geometric representations must also be interpolated. First the geometric representations of P_S and P_T are rendered into two texture-arrays, which are later blended according a factor $\beta \in [0,1]$, which is calculated as follows:

$$\beta = clamp\left(\frac{\sigma - a_{P_S}}{b_{P_S} - a_{P_S}}, 0, 1\right)$$

The interval $[a_{P_S}, b_{P_S}]$ defines in which section of the curve interpolation the geometric representations should be blended.

4 AUTHORING WORKFLOW

Our system supports interactive editing of the complete deformation curve parameterization and preset configuration at run time. To create a visualization, the user has to perform two steps: 1) adjust global settings required for every preset and 2) create or modify presets. According to Section 3.3 the user is required to select the number of control points and set the global number of tag points l, which are equally distributed over the length of the curve initially. This defines the number of styling sections implicitly.

After the global settings are defined, the user can modify the position and orientation of the virtual camera (ϕ) using standard interaction metaphors and edit the deformation curve parameters using direct manipulation of the curve control points. Further, the tag points can be moved along the deformation curve (which alters u) and the size of transition zone between two sections can be adjusted by altering δ. The user directly manipulates the tag points and the B-Spline control points using an interactive 2D widget (inset in Fig. 5). The scene models $\mathscr{G}$ can be loaded and assigned to the respective styling sections by dragging a geometric representation instance G to a respective styling section S. If the geometric representations of the different presets should not be interpolated over the complete interpolation interval, the user can adjust the parameters a and b. Finally, the start s and the end e parameters may be adjusted. These steps are then repeated for every preset.

Once all presets are prepared, the user can fine tune ϕ and τ to achieve the desired transitions. In terms of authoring effort, none of the depicted visualizations took more than three minutes to prepare. In all cases, the most time-consuming steps were the fine-tuning of the transition behavior and the modulation of the blending between the styling sections.

5 INTERACTIVE RENDERING

Our interactive visualization prototype is based on multi-pass rendering using OpenGL and OpenGL Shading language (GLSL). During multi-pass rendering, for each section the global space deformation is applied in the vertex shader. Each deformed geometric representation is written to an off-screen buffer, using Render-To-Texture (RTT) [16]. Finally the textures are composed. Details on the implementation are given in this section.

5.1 Global Deformation Computation

As described in Section 3.2 the deformation can be subdivided into two steps. First, every vertex V is aligned parallel to the camera viewing angle ϕ_a. To achieve this the viewing angle is recomputed on a per frame basis and the according rotation matrix $\mathbf{R}_A$ is passed to the vertex shader. Multiplying V with $\mathbf{R}_A$ yields V', which is projected on the reference plane R. Its initial distance d is stored in a shader variable. Second, the control point and tangent vector of the B-Spline curve is evaluated per vertex, to setup $\mathbf{M}_{C(t)}$. One possibility is to evaluate the B-Spline in the vertex shader. This implies, that the specific formulas to evaluate the parametric curves are known at compilation time and are fixed in the vertex shader code. A change of the parametric curve would lead to a change of the shader code. Instead, we decided to compute the position and tangent vector of the B-Spline curve off-line on the CPU. Thus, the B-Spline curve must be evaluated once a frame instead of once a vertex.

As mentioned in Section 3.2 the B-Spline must be arc-length parametrized. The lookup table is precomputed on the CPU and passed to the vertex shader, for the composition of styling sections, using a 32bit luminance texture. The texture lookup is performed by the parameter t, yielding the arc-length corrected values. The quality of the arc-length approximation depends on the number of precomputed samples. The bilinear interpolation during texture filtering provides a second parameter interpolation. This enables us to reduce the number of samples, without loosing precision. Experiments have shown that 2000 samples are sufficient for an arc-length preserving parametrization.

During the algorithm for arc-length parameterization we further compute the corrected position and tangent vectors of the B-Spline curve on the CPU. These values are stored in a texture that is later used as a lookup table in the vertex shader. The 2D-vectors $C(t)$ and $C'(t)$ are encoded in a 32-bit RGBA texture. The lookup table must be recomputed, if the setup of the parametric curve, e.g. the number or the position of the control points, changes. Thus, for a static curve setup, e.g. the user does not change the viewing angle of the virtual camera, no overhead is introduced. During view-dependent preset interpolation, the lookup table may be updated once per frame.

5.2 Compositing of Styling Sections

The composition consists of two steps: (1) Multipass RTT and (2) image-based composition in the fragment shader. To compose the potential different geometric representations of P_S and P_T, we choose an image-based compositing method, because it is generic and does not require knowledge of the underlying geometric representation. Every styling section of the presets is rendered into separate textures using RTT. Each texture contains RGBA information at viewport resolution. During rendering, a fragment shader adjust the α-value of a fragment according to the styling section boundaries defined by T_i and T_{i+1}, so that:

$$\alpha = \begin{cases} 1 & u_{T_i} + \delta_{T_i} \leq t \leq u_{T_{i+1}} - \delta_{T_{i+1}} \\ \frac{(u_{T_{i+1}} + \delta_{T_{i+1}}) - t}{2 \cdot \delta_{T_{i+1}}} & u_{T_{i+1}} - \delta_{T_{i+1}} < t \leq u_{T_{i+1}} + \delta_{T_{i+1}} \\ 0 & otherwise \end{cases}$$

After RTT is performed, the $2 \cdot (l-1)$ textures ($l-1$ textures per preset) are blended into the frame buffer. The blending of the layers is performed as follows: The first $(l-1)$ textures, encoding P_S, are blended based on α starting with the most distant styling section. The resulting fragment color is temporally stored. This procedure is repeated for the styling sections of P_E. Finally the two colors are blended based on β (see Section 3.5).

In addition thereto, Fig. 6 shows an application examples of the used stylization algorithms. In a preprocessing step, we compute light maps (ambient occlusion term only) for the complete model. At runtime, during the compositing step, we apply edge-detection based on normal and depth information of a fragment and we further unsharp-mask the depth buffer [11] to improve the perception of complex scenes by introducing additional depth cues.

6 RESULTS & DISCUSSION

6.1 Application Examples

We have tested our visualizations using different data sets. Besides photo realistic 3D city models, our ap-

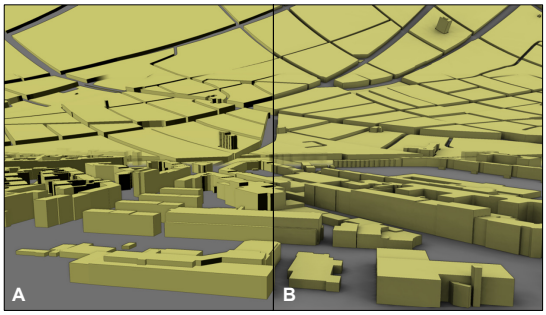


Figure 6: Comparison of applied stylization techniques for generalized virtual city models. A: Directional lighting and edge-enhancement. B: Precomputed ambient occlusion and edge-enhancement.

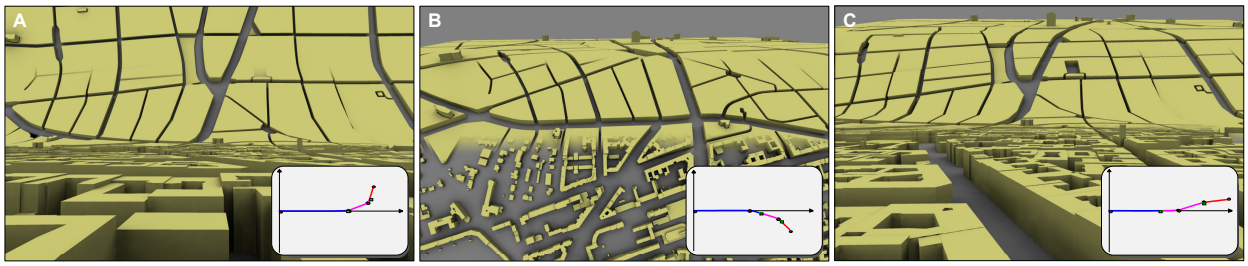


Figure 7: Exemplary visualization using B-Spline curves with six control points to enable hard transitions between three planar regions.

proach is in particular suitable for the depiction of different versions of generalized city models [8] (Fig. 2, Fig. 7). Despite the reduction of geometric complexity, the cell-based generalization also reduces the cognitive load of the user by displaying higher levels of abstraction. In comparison to the map-based stylization (Fig. 5), the generalized geometry is less expressive. The geometry must be enhanced, e.g., with labels, or textures, to communicate additional information to the user. Further, we use two model versions of the Grand Canyon with 524,288 triangles each. The first version uses a height-map as well as an aerial image, while the second version represents a flat terrain with a tourist map applied. Fig. 5 shows the application of the model with a grid applied to emphasize the deformation.

During our experiments, we observed that the usage of more than three styling sections is rather distracting than informative to the user. A high number of sections also reduces the available space for each section. Thus, the amount of objects that can be visualized within a single section decreases. A similar effect arises if the transition zone between two sections (controlled by δ) is chosen to large. Further, the interval $[a_{P_S}, b_{P_S}]$, which control the blending of the geometric representations of P_S and P_T, should be set to initiate the blending briefly after the beginning or before the end of the curve interpolation.

To have a good control over the view-dependent behavior of the global deformation three visualization presets are sufficient, e.g., for a low, a medium and a high viewing angle. To gain more control or to fine tune the interpolation behavior we recommend to use more visualization presets.

6.2 Preliminary User Evaluation

We performed a preliminary user evaluation with 44 participants. The task is to navigate along a route with the help of a static image from a mobile navigation device. Therefore, we prepared 10 routes with a different complexity that partially contained landmarks. For each route we generated 4 visualizations using different perspectives: (1) orthographic (2D), (2) central (3D),(3) progressive and (4) degressive perspective. We presented the participants 26 image pairs. Each pair depicted the same route using two different perspectives. The user were asked which visualization they favor.

The results show that 80,7% of the participants favor the orthographic perspective instead of a central perspective. This is reasonable since a 2D map is a very established mean for navigation. Furthermore we observed that 76,1% prefer the degressive perspective instead of a central perspective. This indicates a demand for multi-perspective views for navigation. With our technique it becomes possible to combine the progressive perspective for a low viewing angle with the orthographic perspective for large viewing angles and thus provide the benefits of both visualization in one navigation tool.

6.3 Performance Evaluation

The performance tests are conducted using a NVIDIA GeForce GTX 285 GPU with 2048 MB video RAM on a Intel Xeon CPU with 2.33 GHz and 3 GB of main memory. The tests are performed at a viewport resolution of 1600×1200 pixels. Table 1 shows the results of our performance evaluation. All models are rendered using in-core rendering techniques with 8 $\times$ anti-aliasing. The performance mainly depends on the number of tag sections, thus the number of required rendering passes, and the geometrical complexity of the scenes attached to them. Due to the heavy usage of render-to-texture in the compositing steps, the performance also depends on the size of viewport. Here, the additional amount of graphics memory $O(l)$ required for a number of global styling section l can be estimated by: $O(l) = 2 \cdot l \cdot w \cdot h \cdot 4 \cdot p$ bytes. Our prototype uses a precision $p = 2$ byte per channel, which is sufficient for post-processing stylization.

Table 1: Comparative performance evaluation for different test scenes (in frames-per-second). The abbreviation LoA 0/1 names the configuration of a preset with two different models (LoA 0 and LoA 1) assigned to the two styling sections.

Preset config.	#Vertex	#Face	FPS
LoA 0/1	1,219,884	477,437	21
LoA 1/2	380,689	364,500	39
LoA 0/1/2	1,443,895	720,587	17

6.4 Limitations and Future Work

The presented approach implies a number of conceptual limitations. First, the number of control and tag points must be the same for each preset in a visualization. Further the visual quality of our approach relies on a sufficient vertex density of the geometric representations. We strive towards the application of hardware tessellation shader units to ensure this property for general scene geometry. Furthermore, the rendering concept is not optimized. At the moment each styling sections requires a single rendering pass. If two or more styling sections contain the same geometric representation, they can be treated as one single styling section reducing the number of rendering passes. The same applies for the geometry interpolation. The number of vertices can be further reduced by a culling algorithm based on the boundaries of the styling sections.

7 CONCLUSIONS

This paper presents a concept and interactive rendering technique for view-dependent global deformations that can be used for the effective visualization of 3D geovirtual environments, such as virtual 3D city and landscape models. It presents an approach for a view-dependent parameterization and interpolation of global deformations based on B-Spline curves. The application of such parametrized curves offers the possibility to customize or extend traditional perspectives, e.g. degressive or progressive perspectives, in a comprehensible and flexible way. Further, the definition of camera-dependent presets and their automatic interpolation overcomes the restriction of existing multi-perspective visualization. In addition, we provide a concept for assigning different geometric representations to specific sections of a curve, which offers more freedom of design. We further present a prototypical implementation that enables hardware-accelerated realtime image synthesis as discussed in our performance evaluation.

ACKNOWLEDGMENTS

This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group "3D Geoinformation". The authors like to thank Tassilo Glander for providing the data sets of the generalized city model of Berlin and Haik Lorenz for his support and critical comments.

REFERENCES

[1] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Proc. of the EG Workshop on Rendering Techniques*, pages 125–136, 2000.

[2] Alan H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84*, pages 21–30, New York, NY, USA, 1984. ACM.

[3] Heinrich Caesar Berann. The world of h.c. berann. web site.

[4] John Brosz, Faramarz F. Samavati, M. T. Carpendale Sheelagh, and Mario Costa Sousa. Single camera flexible projection. In *NPAR '07*, pages 33–42, New York, NY, USA, 2007. ACM.

[5] Nicholas Burtnyk, Azam Khan, George Fitzmaurice, Ravin Balakrishnan, and Gordon Kurtenbach. Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls. In *UIST '02*, pages 101–110, New York, NY, USA, 2002. ACM.

[6] Patrick Degener and Reinhard Klein. A variational approach for automatic generation of panoramic maps. *ACM Trans. Graph.*, 28(1):1–14, 2009.

[7] Martin Falk, Tobias Schafhitzel, Daniel Weiskopf, and Thomas Ertl. Panorama maps with non-linear ray tracing. In *GRAPHITE '07*, pages 9–16, New York, NY, USA, 2007. ACM.

[8] Tassilo Glander and Jürgen Döllner. Abstract representations for interactive visualization of virtual 3d city models. *Computers, Environment and Urban Systems*, 33(5):375 – 387, 2009.

[9] Markus Jobst and Jürgen Döllner. Better perception of 3d-spatial relations by viewport variations. In *VISUAL '08*, pages 7–18, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Haik Lorenz, Matthias Trapp, Jürgen Döllner, and Markus Jobst. Interactive multi-perspective views of virtual 3d landscape and city models. In *AGILE Conf.*, pages 301–321, 2008.

[11] Thomas Luft, Carsten Colditz, and Oliver Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Trans. Graph.*, 25(3):1206–1213, 2006.

[12] D. Martín, S. García, and J. C. Torres. Observer dependent deformations in illustration. In *NPAR '00*, pages 75–82, New York, NY, USA, 2000. ACM.

[13] Sebastian Möser, Patrick Degener, Roland Wahl, and Reinhard Klein. Context aware terrain visualization for wayfinding and navigation. *Computer Graphics Forum*, 27(7):1853–1860, 2008.

[14] Qunsheng Peng, Xiaogang Jin, and Jieqing Feng. Arc-length-based axial deformation and length preserved animation. In *CA '97*, page 86, Washington, DC, USA, 1997.

[15] John W. Peterson. Abstract arc length parameterization of spline curves.

[16] Matt Pharr and Randima Fernando. *GPU Gems 2*. Addison-Wesley Professional, 2005.

[17] Simon Premoze. Computer generated panorama maps. In *ICA Mountain Cartography Workshop*, 2002.

[18] Huamin Qu, Haomian Wang, Weiwei Cui, Yingcai Wu, and Ming-Yuen Chan. Focus+context route zooming and information overlay in 3d urban environments. *IEEE TVCG*, 15(6):1547–1554, 2009.

[19] Paul Rademacher. View-dependent geometry. In *SIGGRAPH '99*, pages 439–446, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[20] Richard Franklin Riesenfeld. *Applications of b-spline approximation to geometric problems of computer-aided design.* PhD thesis, Syracuse, NY, USA, 1973.

[21] David F. Rogers. *An Introduction to NURBS: With Historical Perspective*. Morgan Kaufmann, 2000.

[22] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH*, 20(4):151–160, 1986.

[23] Karan Singh. A fresh perspective. In *Proc. Graphics Interface*, pages 17–24, May 2002.

[24] Shigeo Takahashi, Naoya Ohta, Hiroko Nakamura, Yuriko Takeshima, and Issei Fujishiro. Modeling surperspective projection of landscapes for geographical guidemap generation. *Computer Graphics Forum*, 21:2002, 2002.

[25] Shigeo Takahashi, Kenichi Yoshida, Kenji Shimada, and Tomoyuki Nishita. Occlusion-free animation of driving routes for car navigation systems. *IEEE TVCG*, 12:1141–1148, 2006.

[26] Scott Vallance and Paul Calder. Multi-perspective images for visualisation. In *VIP '01*, pages 69–76, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

A caching approach to real-time procedural generation of cities from GIS data

Brian Cullen
Trinity College Dublin
cullenb4@cs.tcd.ie

Carol O'Sullivan
Trinity College Dublin
Carol.OSullivan@cs.tcd.ie

ABSTRACT

This paper presents a method for real-time generation of detailed procedural cities. Buildings are generated as needed from real GIS data, using modern techniques that can generate realistic content and without having a huge impact on the rendering system. The system uses a client-server approach allowing multiple clients to generate any part of the city the user wishes without requiring the full data-set, or any pre-generated models. The paper introduces the use of object oriented shape grammars to reduce redundant code and presents a parallel cache to allow real-time generation of detailed cities.

Keywords: Procedural Modelling, GIS Data, Buildings, Cities, Real-Time Rendering.

1 INTRODUCTION

Procedural modelling of urban environments has become an important topic in computer graphics. With the ever increasing demand for larger and more realistic content in games and movies, the time and cost to model urban content by hand is becoming unfeasible. Apart from the entertainment industry, large urban models are also desired for urban planning applications and emergency response training.

We present a client-server system capable of generating huge cities of any size without requiring the client to download large 3d geometrical data sets. Our main contributions are as follows:

1. We propose the use of object oriented shape grammars to combat redundancies when creating buildings with multiple different styles.

2. We introduce a multi-state parallel cache that procedurally generates the city's geometry before it becomes visible. We will demonstrate frame-rate improvements over a system that simply generates buildings as they are needed.

While many cache based approaches have been proposed for rendering large terrains, the use of such techniques has not been explored for procedural generation of urban models. Numerous problems occur as rendering the buildings takes much less time than generating them. We aim to tackle this problem with a simple solution that can be used with existing techniques for terrain paging.

After an overview of our system and how we can utilise GIS data to model real cities (Section 3), we then introduce the idea of object oriented shape grammars (Section 4) demonstrating how they can be used to make simple changes to a building without creating redundant code. In Sections 5 and 6 we present our cache based system that can generate huge cities in real-time with interactive frame-rates and evaluate it. Example code of object oriented shape grammars is listed in the Appendix for the interested reader.

2 RELATED WORK

This section will review current techniques for the procedural generation of 3d building models. We will mainly review systems that employ production systems as they have been the most successful at generating realistic content. Other approaches based on stochastic texture synthesis ideas are touched upon briefly.

Detailed architectural models can be created using production systems (a set of symbols that are iteratively replaced according to a well defined grammar) but require a modeller to manually write rules. Their strength lies in the ability to provide detailed descriptions and yet randomness in a structured way.

Parish et al. [24] introduced the idea of using L-Systems [26] to model architectural content. L-Systems are production systems that use the parallel replacement of symbols in a string to simulate a growth process. L-Systems have previously achieved a lot of success in modelling trees and plants [27, 23], but have limitations in modelling buildings (since a building structure is more spatially constrained and does not reflect a growth process).

Stiny pioneered the idea of shape grammars [33, 31, 32] which can be used for generating complex shapes within a given spatial area. Shape grammars have been used for the construction and analysis of architectural designs [5, 8, 34, 12]. However, Stiny's original shape grammar operates on sets of labelled points and lines

and is difficult to implement on a machine because of the number of transformations that must be searched before a rule can be selected and applied.

Wonka et al. [37] modify the idea of shape grammars to better represent building facades. They use a split grammar in which building facade is derived using a sequence of split and repeat commands to subdivide a planar shape.

Müller et al. [21] expand on this idea by developing the CGA shape grammar. This grammar includes environmental parameters that allow a shape (a part of a derived facade) to query if it is occluded by something else in the city, thereby aiding the placement of windows and doors. CGA shape is continually being improved and has even been used to reconstruct archeological sites [22] and is used in commercial products like CityEngine [1].

Recently Kracklau et al. [13] presented a new generalised language based on Python. They can create powerful descriptions by passing non-terminals as parameters, thus enabling abstract templates to be defined.

Shape grammars alone are not sufficient to generate realistic roofs on buildings. Laycock et al. [14] demonstrate a technique to generate roof models in different styles from a building footprint. They modify the straight skeleton algorithm proposed in [7] to generate different roof types. Soon [30] describes an algorithm capable of modelling roofs common to east Asian buildings, like temples and pagodas.

A completely different approach to production grammars takes concepts from texture synthesis and applies them to 3D models. Texture synthesis traditionally extrapolates image data by incrementally adding bits of the image that best match a small neighbourhood. This can produce very convincing results [35, 6, 15].

Merrell and Manoch [18, 19, 20] present a method that takes an example model as input and can produce larger models that resemble it. Output models are still very random and lack the fine control that production systems provide. Synthesis based approaches to generating new models are still very slow and are thus not applicable for interactive applications.

Layout generation concerns the automatic layout of roads and placement of urban content that is crucial for generating an entire city. Urban planning applications require the possibility to view changes to city layouts and to see the effect a proposed road network would have on traffic congestion. Using procedural techniques, such changes can be made interactively which is a great improvement over manual systems.

Parish et al. [24] introduce the use of L-Systems to grow road networks in a similar way to branches on a tree. This was one of the corner-stone papers in the area of procedural cities. However, it is difficult to fine tune the results because the variables do not give enough control over the road layout.

Chen et al. [4] introduce the use of tensor fields to guide road network generation. The user edits the tensor fields using interactive techniques discussed in [38]. Users can then interactively edit individual roads in a quick and easy manner.

Aliaga et al. [3] take a different approach to reconfiguring road networks. Using vector data of roads they form a graph to represent road intersections and parcels of land. Then, using k-means clustering [17], user-deformed parcels are replaced with similar parcels from elsewhere in the city. In [2] they improve on this system to allow the synthesis of completely new areas of the city. Cities with different road structures can then be blended together.

Grueter et al. [9] use a lazy generation technique to construct a potentially infinitely large city. Buildings are constructed when they are visible in the view frustum. The system seeds a random number generator based on the building's coordinates, thereby allowing each building to maintain a persistent style. Whelan et al. [36] present a system that allows real-time interaction in modifying roads and tweaking parameters. The user provides a height map and lays the roads, after which the system automatically places buildings and other details. The buildings are simple extrusions with texture and bump maps. Recently Haegler et al. [10] presented a system capable of generating detailed cities in real-time by carrying out procedural generation on the GPU.

Cache based techniques have been used extensively in real-time rendering. Paging is a popular technique for rendering large terrains [28, 16, 39]. Slater et al. [29] present a caching system that exploits temporal coherency to accelerate view culling. Akenine-Möller et al. [11] discuss many modern real-time rendering techniques including level of detail, batch processing and imposters.

3 SYSTEM OVERVIEW

In this section we present a system that can produce large detailed virtual cities in real-time using GIS data. Previous approaches discussed in Section 2 focus on either pre-generating large cities or are limited to simple grid layouts and building geometry with random styles. The system presented continuously updates the city by streaming GIS data from a server along with style descriptions for every building, without interrupting the rendering system.

Urban GIS is preprocessed and stored in a database along with style descriptions for every building for quick referencing. This preprocessing step is explained in section 3.1. Style sheets that control the facade generation are loaded at run-time and are stored in a hash-table on the client's system. The geometry cache updates itself based on the camera's position in the environment, downloading the surrounding environment

data from the GIS database. This includes the position and shape of building footprints and style parameters (such as texture id, height and style id) used for generating the buildings. This allows persistent generation of the city. The cache controls what geometry is procedurally generated based on its distance from the camera. Meshes for the roads and buildings are then batched together for efficient rendering and sent to the render system. This process is described in detail in Section 5.

3.1 Data Extraction from GIS

The GIS data recorded contains detailed urban planning information, which is stored in different semantic layers that make it easy to access the building layouts. However, since the data is simply represented by a set of poly-lines, it is necessary to determine which lines belong to the same buildings. Figure 1 illustrates this process. The following algorithm describes how to extract the building layouts:

1. Create a graph representing all the vertices and edges.
2. Start at the bottom left node which contains two or more edges.
3. Follow the least interior angle edges until the starting node is reached again, thus creating a cycle.
4. Decrement the degree of every node along the cycle.
5. Repeat from Step 2 until no nodes with a degree greater than one remain.

A similar approach was taken by Pina et al. [25], however, individual buildings are extracted as opposed to urban blocks. The extracted building footprints are then loaded into a database for quick referencing by the system. A similar technique is used to extract the roads and insert the road network graph into a database.

4 BUILDING GENERATION

Buildings are procedurally generated using split grammar rules based on [21]. The rules compose of *subdiv*, *repeat*, *insert*, *extrude*, *detrude* and *comp* commands, which can subdivide and decompose shapes into new ones.

Comp
Breaks a shape down into the lower dimensional shapes it is composed of. For example, a building is broken down into its composing facades;

Subdiv
Subdivides a shape along a given axis;

Repeat
Subdivides a planar shape several times to fit many new shapes of a given width;

Insert
Replaces a planar shape with an external model;

Extrude
Extrudes a planar shape, thereby creating a new volumetric shape;

Detrude
Detrudes a planar shape, thereby creating a new volumetric shape.

Combinations of these simple commands can produce complex architectural geometry, while building roofs are generated using the approach described in [14]. The rules are specified using a script with a parameterised L-System style syntax:

$$Pred : Exp \rightsquigarrow Command(params)\{Successor\} : Prob$$

If the Boolean *Exp*ression evaluates to *true* then *Command* is carried out on the shape with ID *Pred*ecessor and the resulting output shapes are given the ID *Successor*. Multiple rules can be specified for the same *Pred*ecessor and one is chosen at random based on its *Prob*ability value. This allows some variability among generated shapes.

A simple compiler was built to parse the scripts at runtime and generate a hash table of C++ function objects. This allows the script to be applied extremely quickly to new buildings but also allows parameters to be changed at runtime.

4.1 Object Oriented Buildings

The production system presented in [21] contains a lot of redundant code between different building scripts. It is very cumbersome to rewrite entire building specifications just to make a specific change.

We propose the use of object oriented buildings as a solution to this problem. Figure 2 illustrates this idea. Buildings inherit everything from more abstract styles and only respecify certain aspects of the style. This is achieved by encapsulating semantically relevant production rules in labelled blocks. Each block is given a list of variables that can be changed at runtime or respecified by a child style. Code listings to generate the buildings in Figure 2 can be found in the Appendix. Buildings also inherit their parents' elements (i.e., 3D models that are imported and used to replace certain terminal symbols) and can add or remove from their parents' element set. We allow multiple meshes to be specified, corresponding to different levels of detail for the rendering system. Meshes are swapped with different level of detail meshes depending on their distance to the camera. In this implementation of the system the Ogre rendering engine was utilised to manage level of detail swapping and rendering of the scene. The use of object oriented building styles can simplify the writing of new styles and can link building styles together in a meaningful way.

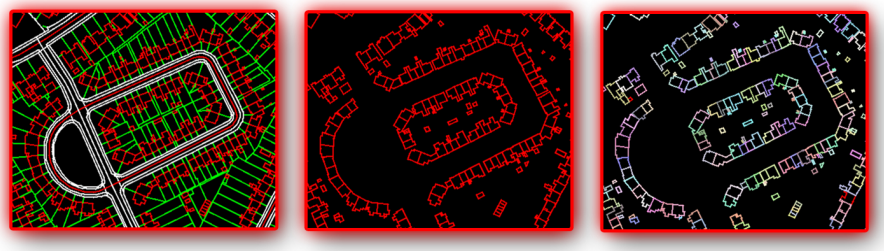


Figure 1: Extracting building footprints from GIS data (left). Layer containing buildings is first chosen by the user (middle), while buildings are then extracted by finding loops in the data (right).

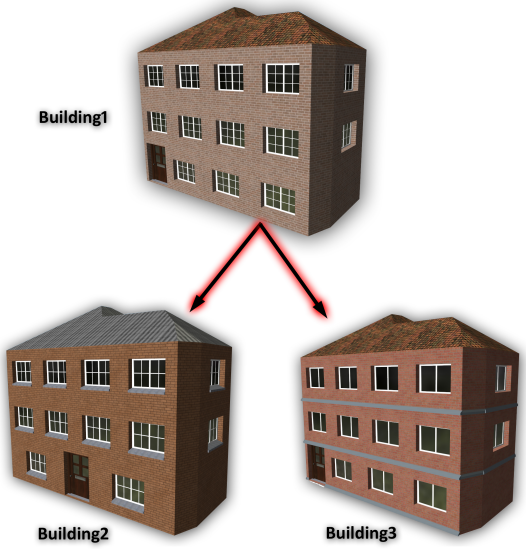


Figure 2: Building2 inherits from Building1, specifying how windowsills should be added. Building3 also inherits from Building1, adding a ledge to each floor. Code listings are provided in the Appendix.

5 REAL-TIME GENERATION

In this section we present our process for generating procedural cities in real-time.

5.1 Parallel Geometry Cache

In order to maintain a constant and high frame rate, building generation should not interrupt the rendering system. We achieve this by introducing a multi-state cache that stores geometry that is currently being generated. The system is based on the idea of paging geometry for rendering large terrains. The world is split into a regular grid as illustrated in Figure 4. The data in the cache has the following three states:

State 1 Geometry descriptions are downloaded from the database and the area is procedurally generated. (Outer white area in Figure 4).

State 2 Meshes are constructed and sent to the graphics card but are not yet rendered (Middle blue area in Figure 4).

State 3 Meshes currently being rendered (Inner green area in Figure 4).

Depending on the camera motion, grid squares that are likely to become visible in the near future are loaded. Geometry descriptions are downloaded from the GIS database, procedurally generated and inserted into the cache. This is done in a separate thread from the rendering system. Only squares that are close to the camera are rendered. If a square is not yet generated, the rendering thread will put it on the end of a queue and try to retrieve the next square.

5.2 Parallel Building Generation

With the trend in computing power drifting towards multi-processor architectures, it is desirable to take advantage of parallel computation. It is possible to procedurally generate multiple buildings at the same time by utilizing parallel processing techniques. Algorithm 3 presents a simple algorithm that can speed up building generation on multiprocessor systems.

```
while NewPage = getPageFromQueue()
  NumBldPerThd = NewPage.NumBlds/NumProc
  for x = 0 to numProccesors-1
      Thread[x] = ForkThread()
      Thread[x].MemoryPool = new MemoryPool
      Thread[x].BuildingList = distBuildings(NumBldPerThd)
      Thread[x].GenerateBuildings()
      NewPage.setBuildingMeshes(Thread[x])
  end for
   SynchroniseThreads()
end while
NewPage.BatchMeshes()
```

Algorithm 3: Algorithm for procedurally generating buildings in parallel.

Each thread maintains a memory pool that is reused for every building it generates, which reduces memory allocation bottlenecks. Threads must synchronise before writing to the cache so that buildings can be batched together for fast rendering.

6 RESULTS

To test the system, we conducted two separate benchmark, which were performed on a machine with the following specifications:

CPU	Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHZ
RAM	4GB
GPU	NVIDIA GeForce 9800GT

First, a frame rate analysis of the system was taken while the camera was moved between two preset points,

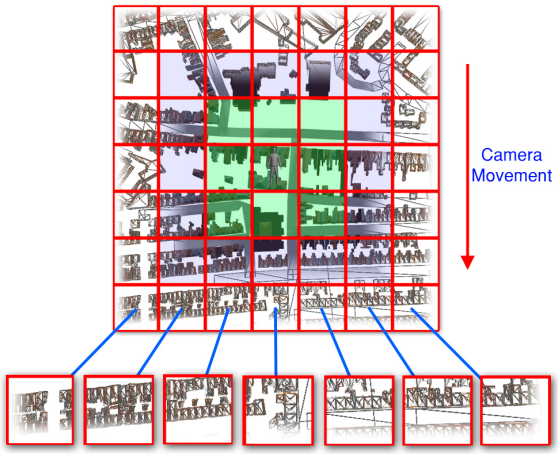


Figure 4: As the camera moves towards the geometry, new pages must be loaded. The pages are organised into a queue and processed in order of their distance to the camera. The buildings within each page should be shared among parallel executing threads.

both with and without the parallel geometry cache (Section 5.1). The results are given in Figure 5. While the camera travelled a distance of 800m in the scene, exactly 2,038 buildings were created. This had a significant effect on the frame rate of the system without a parallel cache. The sudden drops in frame rate correspond with new geometry pages being loaded and cause a jerk in the camera motion. In the system with the parallel cache there is much less jerking when pages are loaded and the overall frame rate stays within acceptable levels.

The second experiment performed was a multi-threaded processing benchmark. Four pages were generated consisting of 10, 100, 1000 and 10,000 buildings respectively. Processing time was logged for each of the pages with building generation distributed over different number of threads. The average results over ten repetitions are shown in Figure 6. A configuration with two threads running in parallel yielded the best performance on the dual core machine. Running the experiment with more threads than processors led to worse results because of the overhead of thread switching. However, this result suggests better performance could be achieved with a greater number of processing cores. Better results were obtained using larger page sizes with 10,000 buildings leading to a 27.48% increase in performance (We suspect that this is due the initial memory pool allocation assigned to each thread). Table 1 shows the number of buildings generated per second for the 10,000 building page test. Each building was set to be strictly the same shape, contained an average of 980 vertices and required 610 shape operations to generate.

Figure 7 demonstrates the type of architecture and scale of the city generated in the tests.

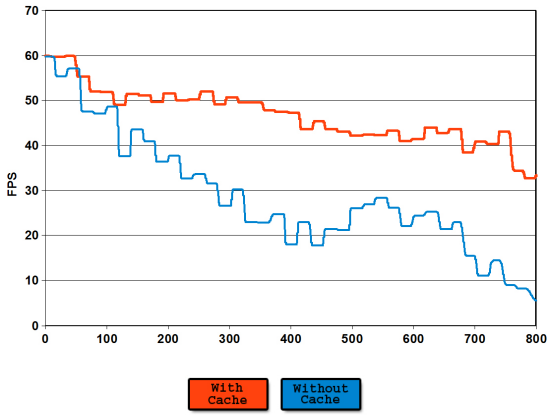
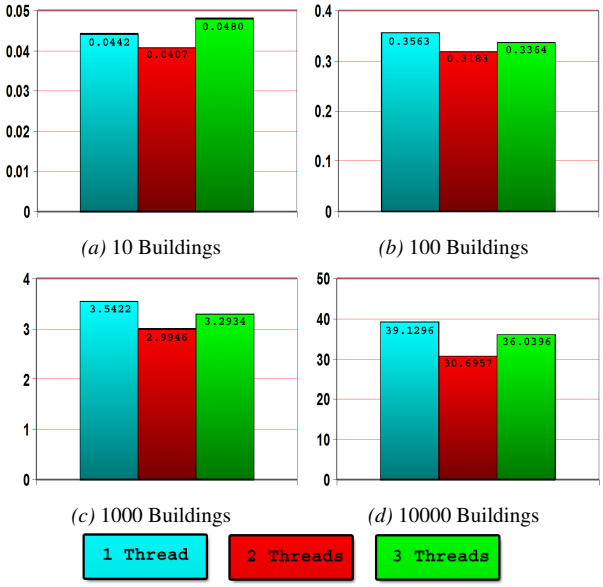


Figure 5: A comparison of results with and without the cache described in Section 5.1. The system with the cache has a much higher frame rate and less jerky movements of the camera. There was an average of 1,922 buildings in the scene at any time with 2,038 buildings created and destroyed over the distance.



(a) 10 Buildings *(b)* 100 Buildings

(c) 1000 Buildings *(d)* 10000 Buildings

Figure 6: Time in seconds to generate buildings with different levels of multithreading. On the dual core machine two threads yielded the best performance.

	Bld/Sec	Ops/Sec	Percent Increase
1 Thread	255.56	15,586.34	
2 Threads	325.78	19,868.86	27.48%
3 Threads	277.47	16,922.73	8.57%

Table 1: Benchmark of multi-threaded processing on the 10,000 building data set. Results are shown for the number of buildings generated per second, the number of shape operations (discussed in Section 4) performed per second and the percentage performance boost over a single threaded configuration.

Figure 7: Output of the system

7 CONCLUSION

We have presented a system that can generate large virtual cities with detailed buildings in real-time. The system can be run over a network while allowing multiple clients with only one data set. We introduced the idea of object oriented building styles that can help reduce code redundancies and make it easier to specify multiple building styles. We also presented a set of benchmarking statistics calculated with different configurations of the system. The results showed that our parallel cache offers superior performance to that of a system that simply generates the buildings as they are needed. We also showed a performance benefit when utilising parallel generation on multi-core processors.

Regarding limitations, currently the system only generates buildings within a single page in parallel. The results from our experiment suggest that improved performance could be achieved by generating sets of pages in parallel, thus handling more buildings per thread and requiring less thread synchronisation. Rendering of the system could improved by implementing occlusion culling and better LOD techniques. In this implementation, different level of details are provided for a building's elements but not the shape of the building itself.

A SHAPE GRAMMAR SYNTAX

In this appendix we present the syntax of our object oriented shape grammar.

The listings correspond to the buildings shown in Figure 2. Semantically relevant production rules can be combined into meaningful blocks. Each block can have its own list of variables that may be changed at runtime. A child class inherits everything from its parents and may redefine a block of rules and its variables. In addition to defining a block of production rules, a class can also define a set of building elements (Listing 9). These elements correspond to terminal symbols in the production system, which should be replaced with external models. A series of meshes can be given to each element specifying a different level of detail. In our system, the distance at which to change a mesh is the same for each element and is specified by the cache system. As with the production rules, probabilities are given for the replacement of terminal symbols with 3D meshes.

```
class Building1 : ElementPack
{
  Footprint{
    FOOTPRINT ~>
      extrude(BUILDING_HEIGHT){ BuildingVol } : 1
    BuildingVol ~>
      comp ("facades"){ FACADE } : 1
  }

  Facade{
    var GroundFloorHeight 1

    FACADE : H > (GroundFloorHeight + 1) ~>
      subdiv("Y",GroundFloorHeight,1r)
      { GROUND_FLOOR | UPPER_FLOORS } : 1
  }

  Ground_Floor{
    var EntranceWidth 0.75
    var DoorDepth 0.1

    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",1r,EntranceWidth,0.1)
      { FLOOR | EntrancePanel |WALL } : 0.3
    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",1r,EntranceWidth,1r)
      { FLOOR | EntrancePanel |FLOOR } : 0.44
    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",0.1,EntranceWidth,1r)
      { WALL | EntrancePanel |FLOOR } : 0.3
    EntrancePanel ~> subdiv("Y",0.02,1r)
      { WALL | Entrance }
    Entrance ~> detrude(DoorDepth)
      { DOOR |WALL } : 1
  }

  Upper_Floors{
    var FloorHeight 1.0

    UPPER_FLOORS ~> repeat("Y",FloorHeight){FLOOR} : 1
  }

  Floor{
    var TileWidth 1.1

    FLOOR ~> repeat("X",TileWidth){TILE} : 1
  }

  Tile{
    var WindowDepth 0.1
    var WindowWidth 0.75
    var WindowHeight 0.5

    TILE ~> subdiv("X",1r,WindowWidth,1r)
      { WALL | Tile | WALL } : 1
    TileC ~> subdiv("Y",1r,WindowHeight,1r)
      { WALL |WindowPlane | WALL } : 1
    WindowPlane ~> detrude(WindowDepth)
      { WINDOW | WALL } : 1
  }
}
```

Listing 8: Listing for simple building

```
class ElementPack
{
  Elements{
    WINDOW:
      "window1LOD1.mesh" "window1LOD2.mesh" : 0.5
      "window2LOD1.mesh" "window2LOD2.mesh" : 0.5

    DOOR:
      "door1.mesh" : 0.2
      "door2LOD1.mesh" "door2LOD2.mesh" : 0.8

    LEDGE:
      "windowLedge1LOD1.mesh" "windowLedge1LOD2.mesh" : 1
  }
}
```

Listing 9: Listing for elements

```
class Building2 : Building1
{
  Tile{
    var LedgeHeight 0.075
    var WLedgeHeight (WindowHeight+LedgeHeight)

    TILE ~> subdiv("X",1r,WindowWidth,1r)
      { WALL | TileC | WALL }: 1
    TileC ~> subdiv("Y",1r,WLedgeHeight,1r)
      { WALL | WindowPlane | WALL }: 1
    WindowPlane ~> detrude(WindowDepth)
      { WindowPlaneInner | WALL }: 1
    WindowPlaneInner ~> subdiv("Y",LedgeHeight,1r)
      { LEDGE | WINDOW }: 1
  }
}
```

Listing 10: Building2 inherits everything from Building1 but specifies how windowsills should be added.

```
class Building3 : Building1
{
  floor{
    var TileWidth 1
    var LedgeHeight 0.075

    FLOOR ~> subdiv("Y",LedgeHeight,1r)
      {LEDGE | FloorU} : 1
    FloorU ~> repeat("X",TileWidth){TILE} : 1
  }
}
```

Listing 11: Building3 inherits everything from Building1 adding a ledge to each floor.

REFERENCES

[1] Procedural inc. - 3D modeling software for urban environments. http://www.procedural.com/.

[2] D. G. Aliaga, B. Beneš, C. A. Vanegas, and N. Andrysco. Interactive reconfiguration of urban layouts. *IEEE Comput. Graph. Appl.*, 28(3):38–47, 2008.

[3] D. G. Aliaga, C. A. Vanegas, and B. Beneš. Interactive example-based urban layout synthesis. *ACM Trans. Graph.*, 27(5):1–10, 2008.

[4] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[5] J. Duarte. *Malagueira Grammar - towards a tool for customizing Alvaro Siza's mass houses at Malagueira*. PhD thesis, MIT School of Architecture and Planning, 2002.

[6] C. Eisenacher, S. Lefebvre, and M. Stamminger. Texture synthesis from photographs. *CGF: Eurographics*, 27(2):419–428, 2008.

[7] P. Felkel and S. Obdrzálek. Straight skeleton implementation. In *SCCG: Spring Conference on Computer Graphics*, page 210–218, 1998.

[8] U. Flemming. More than the sum of parts: the grammar of queen anne houses. *Environment and Planning B: Planning and Design*, 14(3):323–350, 1987.

[9] S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation of 'pseudo infinite' cities. In *GRAPHITE: Computer Graphics and Interactive Techniques*, page 87–ff, New York, NY, USA, 2003. ACM.

[10] S. Haegler, P. Wonka, S. M. Arisona, L. V. Gool, and P. Müller. Grammar-based encoding of facades. *CGF: Eurographics*, 29(4):1479–1487, 2010.

[11] J. Hasselgren and T. Akenine-Möller. PCU: the programmable culling unit. *ACM Trans. Graph.*, 26(3):92, 2007.

[12] H. Koning and J. Eizenberg. The language of the prairie: Frank lloyd wright's prairie houses. *Environment and Planning B: Planning and Design*, 8(3):295–323, 1981.

[13] L. Krecklau, D. Pavic, and L. Kobbelt. Generalized use of Non-Terminal symbols for procedural modeling. *CGF: Eurographics (to appear 2010)*, 2010.

[14] R. G. Laycock and A. M. Day. Automatically generating roof models from building footprints. In *Journal of WSCG*, 2003.

[15] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. In *ACM Trans. Graph.*, pages 541–548, Boston, Massachusetts, 2006. ACM.

[16] Y. Livny, Z. Kogan, and J. El-Sana. Seamless patches for GPU-based terrain rendering. *Vis. Comput.*, 25(3):197–208, 2009.

[17] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[18] P. Merrell. Example-based model synthesis. In *I3D: Symposium on Interactive 3D Graphics and Games*, page 105–112, New York, NY, USA, 2007. ACM.

[19] P. Merrell and D. Manocha. Continuous model synthesis. *ACM Trans. Graph.*, 27(5):1–7, 2008.

[20] P. Merrell and D. Manocha. Constraint-based model synthesis. In *SPM '09: SIAM/ACM Joint Conference on Geometric and Physical Modeling*, page 101–111, New York, NY, USA, 2009. ACM.

[21] P. Müller, T. Vereenooghe, P. Wonka, I. Paap, and L. V. Gool. Procedural 3D reconstruction of puuc buildings in xkipché. In *VAST: Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, page 139–146, 2006.

[22] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[23] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Computer Graphics and interactive techniques*, page 397–410, New York, NY, USA, 1996. ACM.

[24] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[25] J. L. Pina, F. J. Serón, and E. Cerezo. Building and rendering 3d navigable digital cities. In *GI_forum*, pages 167–176, Salzburg, Austria, 2009.

[26] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990.

[27] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. In *SIGGRAPH '01: Computer Graphics and Interactive Techniques*, pages 289–300. ACM, 2001.

[28] J. Schneider and R. Westermann. GPU-Friendly High-Quality terrain rendering. *Journal of WSCG*, 14(1-3):49–56, 2006.

[29] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic caching scheme. In *Department of Computer Science, University College London*, pages 71–78. ACM Press, 1997.

[30] T. T. Soon. Generalized descriptions for the procedural modeling of ancient east asian buildings. In *Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging(CAE'09)*, 2009.

[31] G. Stiny. Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design*, 7(3):343 – 351, 1980.

[32] G. Stiny. Spatial relations and grammars. *Environment and Planning B*, 9(1):113–114, 1982.

[33] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Friedman, editor, *Information Processing '71*, page 1460–1465, Amsterdam, 1972.

[34] G. Stiny and W. J. Mitchell. The palladian grammar. *Environment and Planning B: Planning and Design*, 5(1):5 – 18, 1978.

[35] L. Wei, S. Lefebvre, V. Kwatra, and G. Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Reports, EGSTAR*. Eurographics Association, 2009.

[36] G. Whelan, G. Kelly, and H. McCabe. Roll your own city. In *Digital Interactive Media in Entertainment and Arts*, pages 534–535, Athens, Greece, 2008. ACM.

[37] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.

[38] E. Zhang, J. Hays, and G. Turk. Interactive tensor field design and visualization on surfaces. *IEEE TVCG: Transactions on Visualization and Computer Graphics*, 13(1):94–107, 2007.

[39] Z. Zhou, B. Cai, D. Zhang, and X. Zhang. Paged cache based massive terrain dataset Real-Time rendering algorithm. In *ICIECS: Information Engineering and Computer Science*, pages 1–4, 2009.