ISSN 1213-6972 *Volume 19, Number 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*

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. 3

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.

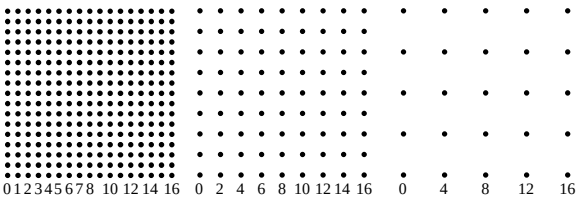


Figure 1. Downsampling initial height map.

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).

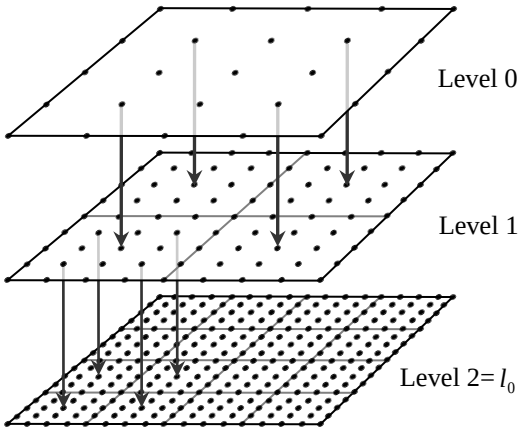


Figure 2. Patch quad tree.

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):

$$\hat{h}_{2i,2j}^{(l)} \in \{\hat{h}_{i,j}^{(l-1)} - 2\delta_l,\ \hat{h}_{i,j}^{(l-1)},\ \hat{h}_{i,j}^{(l-1)} + 2\delta_l\}.$$

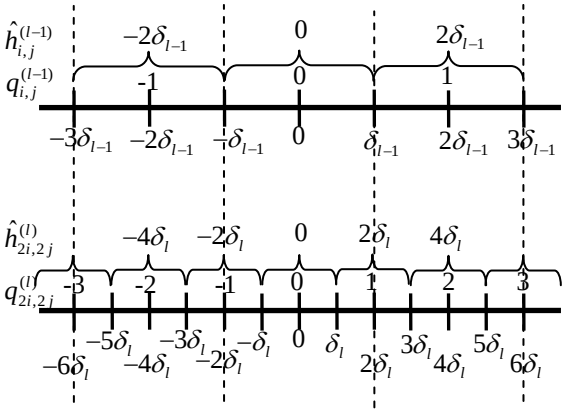


Figure 3. Quantizing two successive levels.

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.

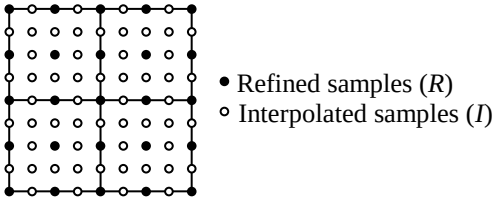


Figure 4. Refined and interpolated samples of the child patches joined height map $\hat{H}_C^{(l)}$.

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

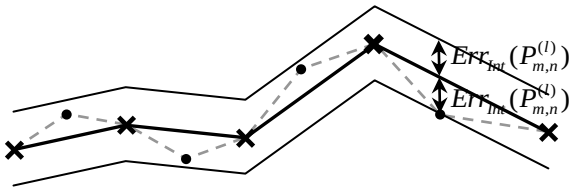
l_0) height map. It is recursively calculated using the same method as used in ROAM [DWS+97] to calculate the nested wedgie thickness:

$$Err_{Appr}(P_{m,n}^{(l_0)}) = 0$$

$$Err_{Appr}(P_{m,n}^{(l)}) = Err_{Int}(P_{m,n}^{(l)}) + \max_{s,t=\pm 1}\{Err_{Appr}(P_{2m+s,2n+t}^{(l+1)})\},$$

$$l = l_0 - 1,..0$$

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).

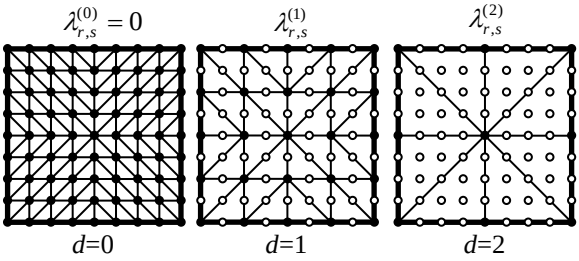


Figure 6. Triangulations of a 9×9 tessellation block.

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:

$$d = \arg\max_d proj(\overline{e_c^{(d),-}, e_c^{(d),+}}) < \varepsilon$$

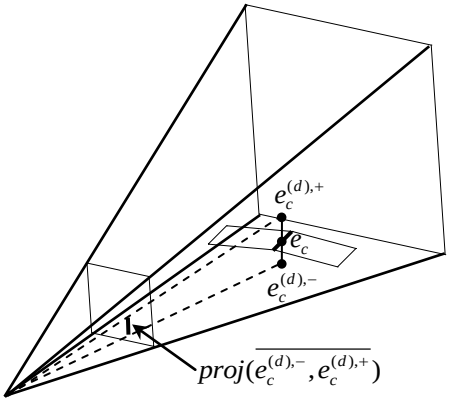


Figure 7. Calculating edge screen space error.

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.

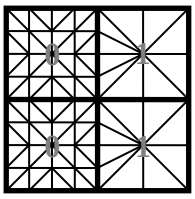


Figure 8. Seamlessly triangulated patch's tessellation blocks.

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.

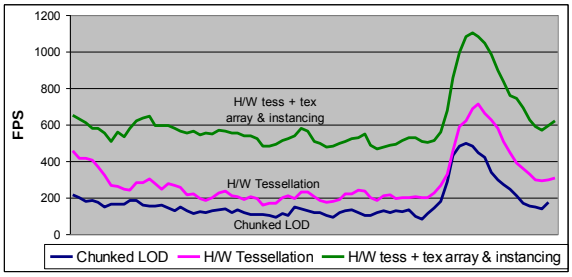


Figure 10. Rendering performance at 1920×1200 resolution.

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.

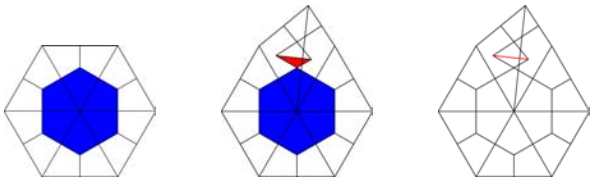


Figure 1: Primal and dual meshes. The left mesh is Delaunay, whereas the other meshes are not Delaunay. The middle mesh shows a dual 0-cell whose area is given by the blue area minus the red area. The red line in the right mesh shows a dual 1-cell with negative length.

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.

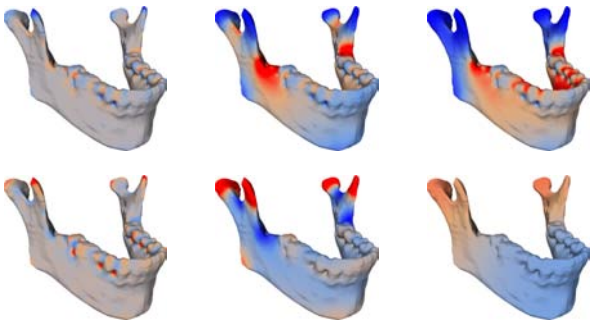


Figure 2: $\operatorname{tr}(k^1(t,p,p))$ (top) and Heat Kernel Signature (bottom) for increasing values of t.

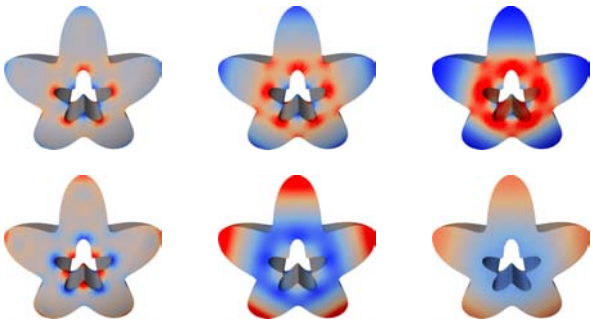


Figure 3: $\operatorname{tr}(k^1(t,p,p))$ (top) and Heat Kernel Signature (bottom) for increasing values of t.

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.

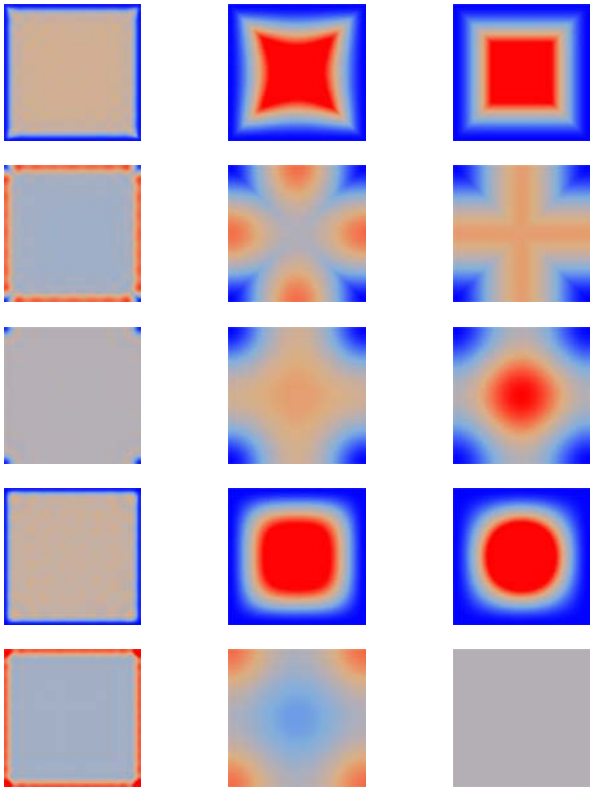


Figure 5: from top to bottom: smaller eigenvalue of $k^1(t,p,p)$, larger eigenvalue of $k^1(t,p,p)$, $\mathrm{tr}(k^1(t,p,p))$, $\det(k^1(t,p,p))$ and Heat Kernel signature for increasing values of t.

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

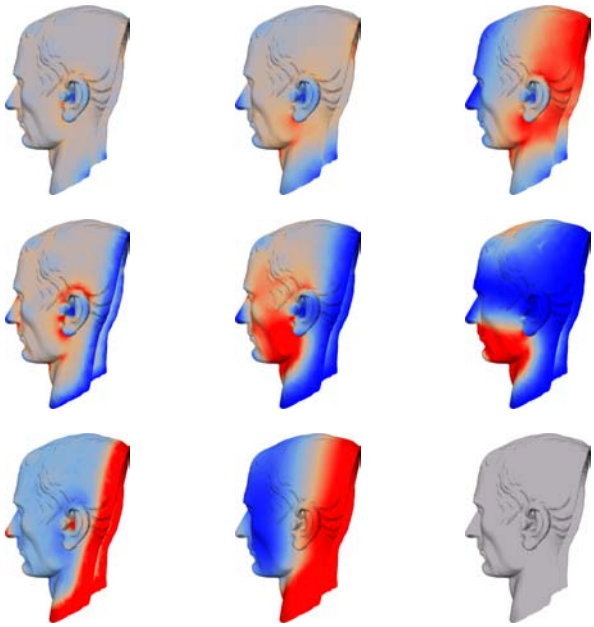


Figure 6: from top to bottom: $\mathrm{tr}(k^1(t,p,p))$, $\det(k^1(t,p,p))$ and Heat Kernel Signature for increasing values of t.

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.

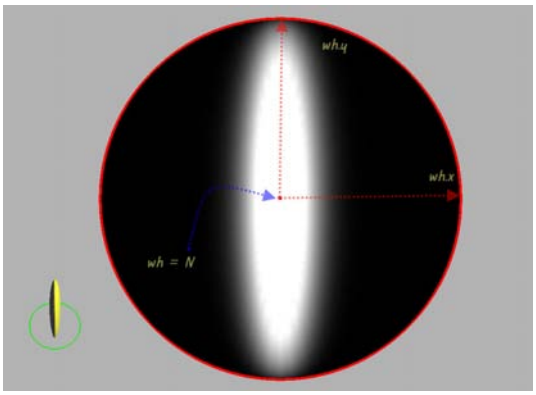


Figure 2: MDF in 2D space

(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-

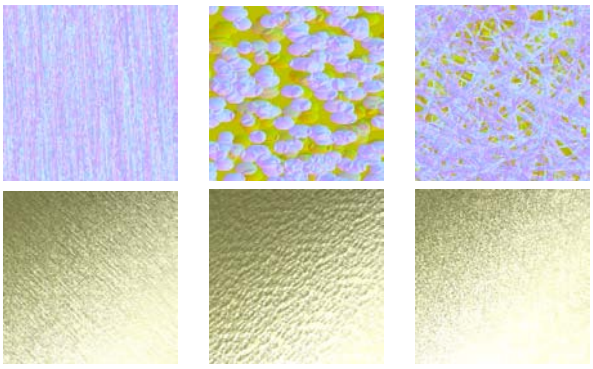
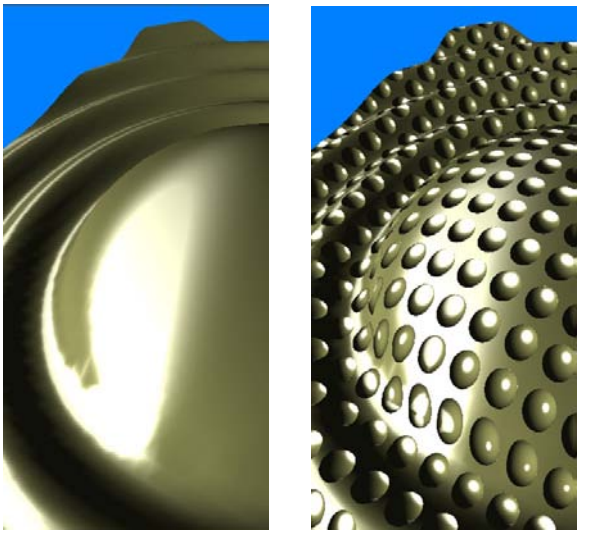


Figure 4: Scratch maps and expected rendering results: (top row) scratch maps and (bottom row) expected results.



(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:

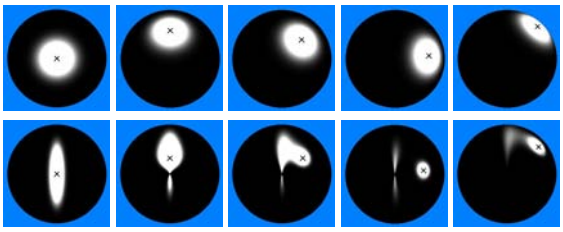


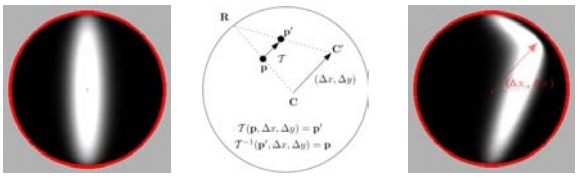
Figure 6: MDF with perturbed normal vectors: (top row) perturbation with isotropic MDF and (bottom row) perturbation with anisotropic MDF.

$$D(\omega_h,\tilde{\mathbf{n}}) = \frac{\sqrt{(e_x+1)(e_y+1)}}{2\pi}(\omega_h\cdot\tilde{\mathbf{n}})^{e_x\cos^2\phi+e_y\sin^2\phi} \quad (2)$$

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.

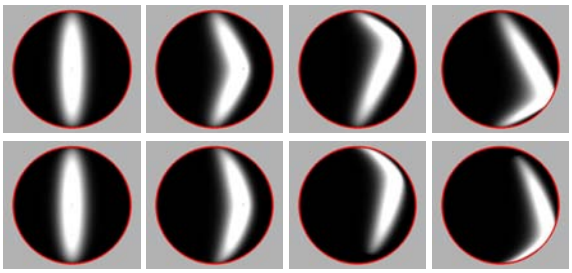


Figure 8: MDF deformation examples: (top row) linear interpolation results and (bottom row) smooth interpolation results.

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:

$$\begin{aligned} D'(\omega_h,\tilde{\mathbf{n}}) = & \\ D(\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}),\mathbf{n}) = & \quad \frac{\sqrt{(e_x+1)(e_y+1)}}{2\pi}\mathscr{T}^{-1}(\omega_h,\tilde{\mathbf{n}}).z^{\varepsilon} \end{aligned} \tag{6}$$

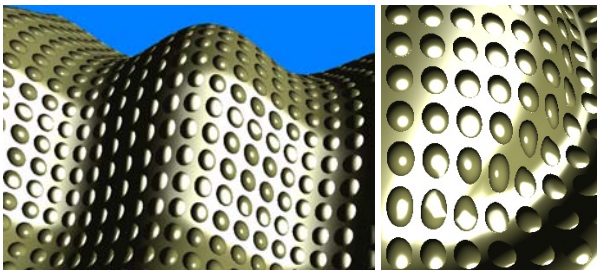
,where the exponent ε is $e_x\cos^2\phi + e_y\sin^2\phi$.

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.

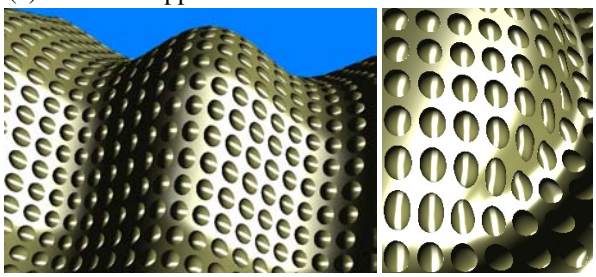
2.4 Scratch Map Generation

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

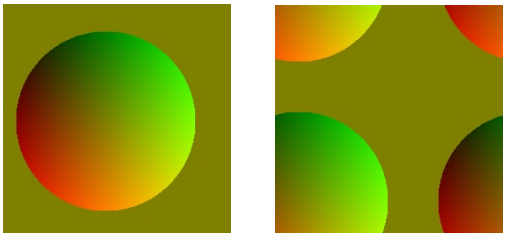


(a) Normal mapped surface without MDF deformation



(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.

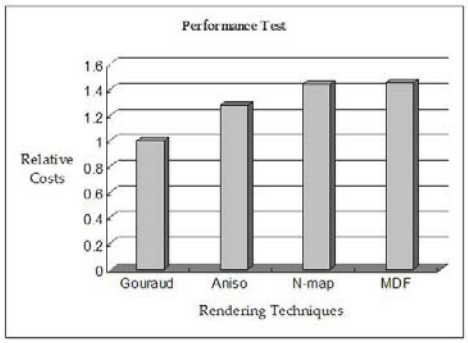


(a) basic pit (b) moved pit

(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

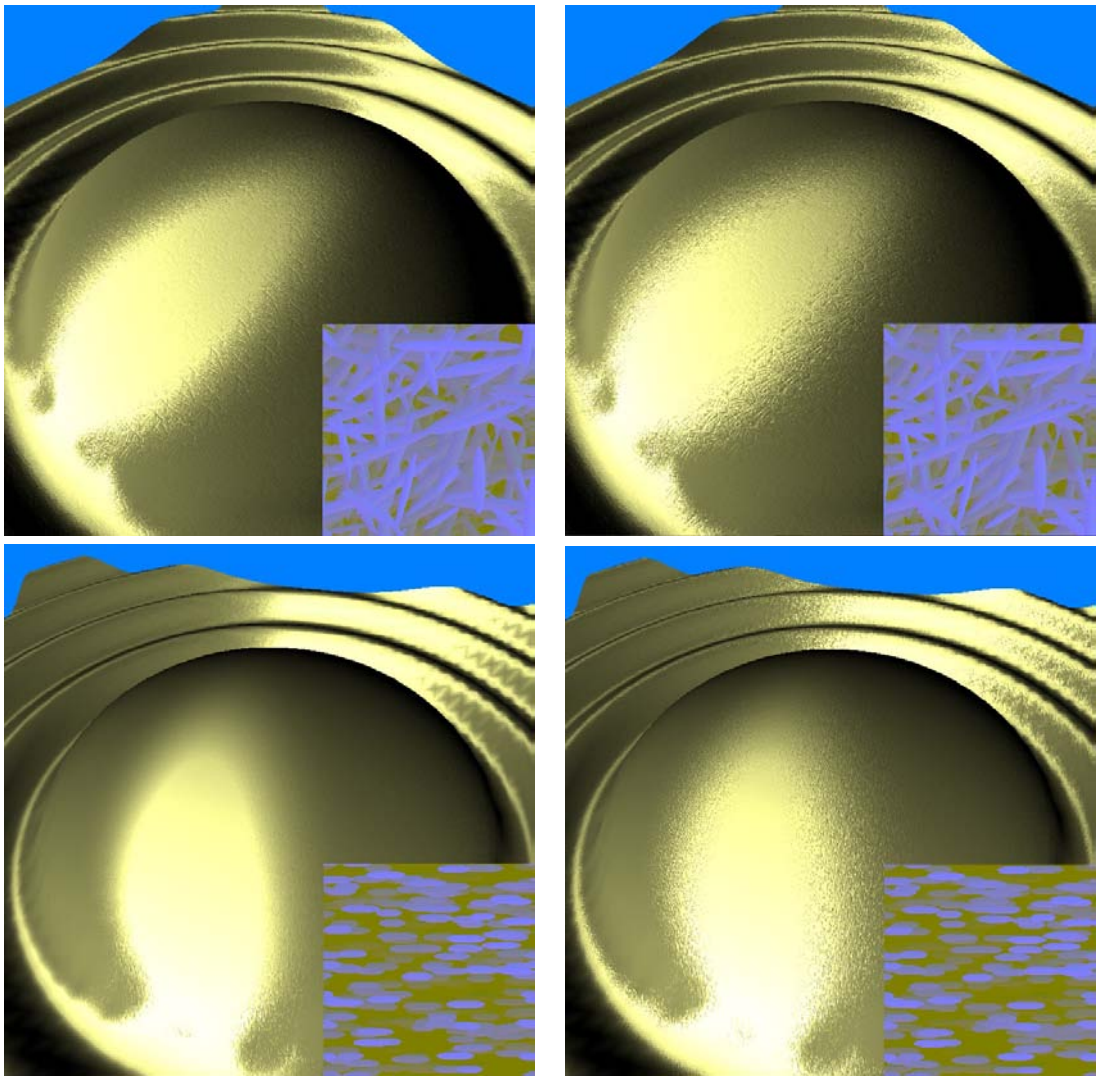
Figure 11: Rendering performance of the proposed method compared with other realtime methods.

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

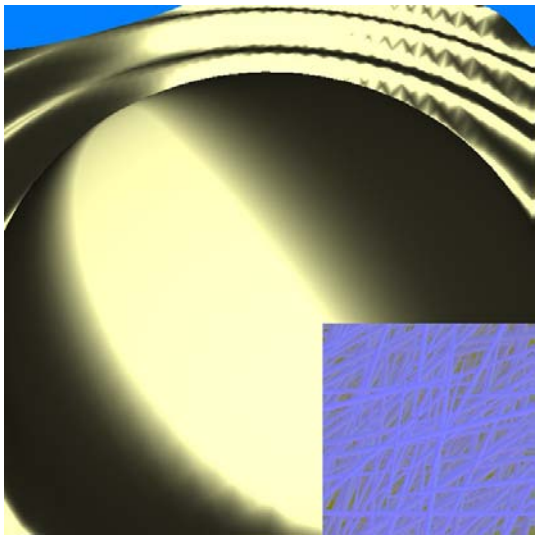
Figure 12: Comparison of light scattering on (a) simple normal mapped surface and (b) normal mapped surface with additional MDF deformation.

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

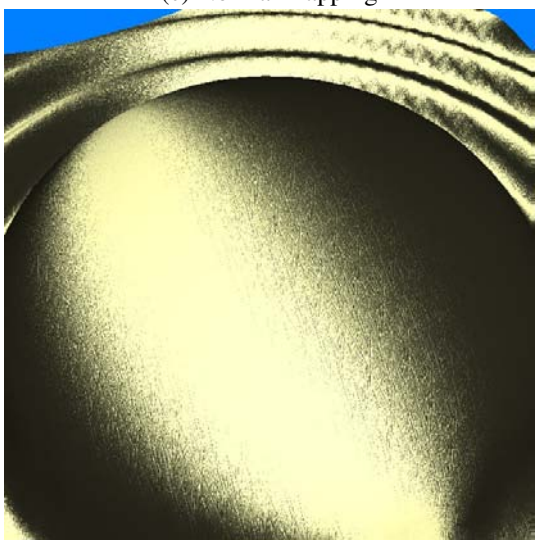


(a) Anisotropic reflection (Ashikhmin-Shirley model)

(b) Anisotropic reflection with environments

(c) Normal mapping

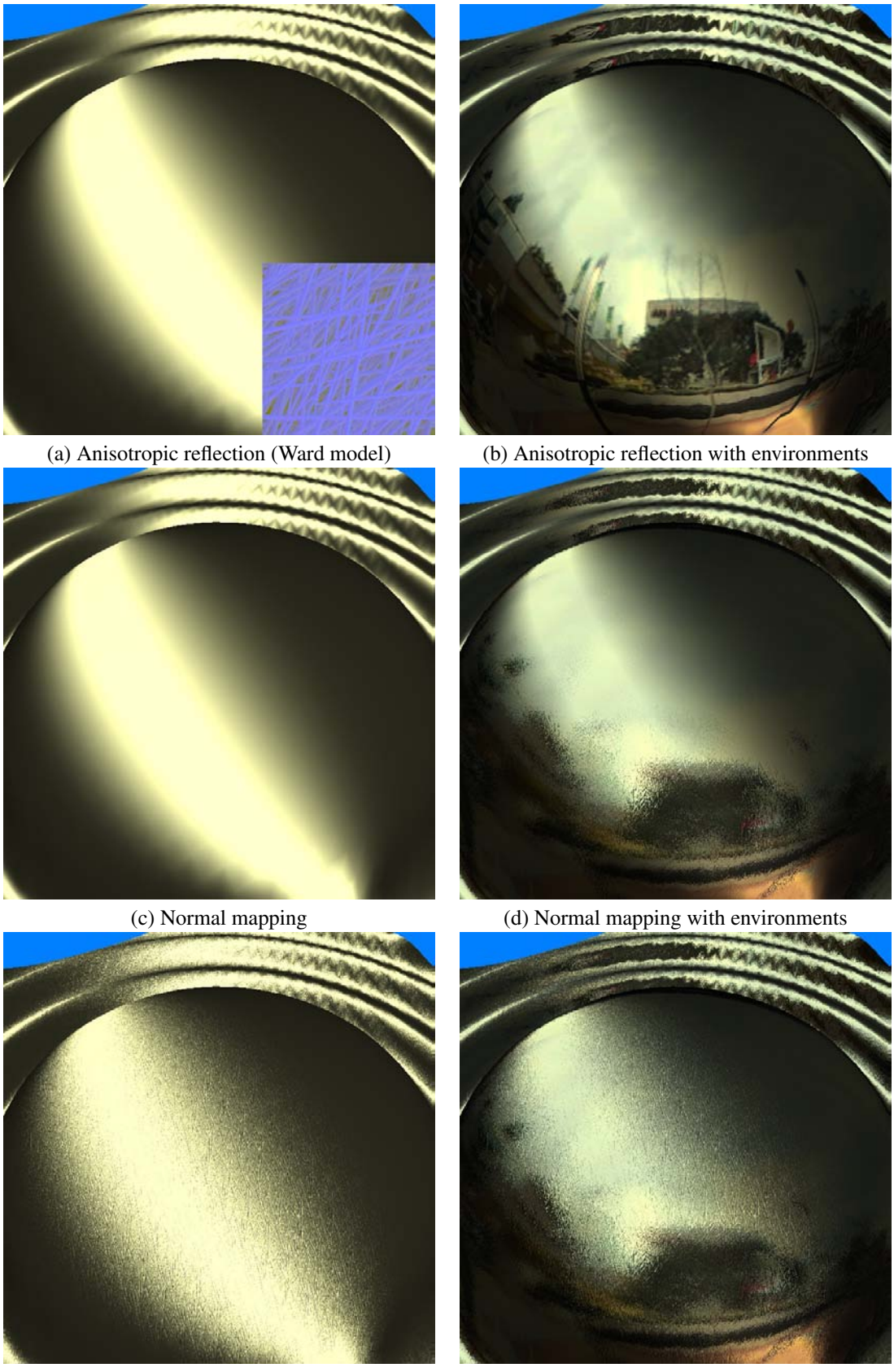
(d) Normal mapping with environments



(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

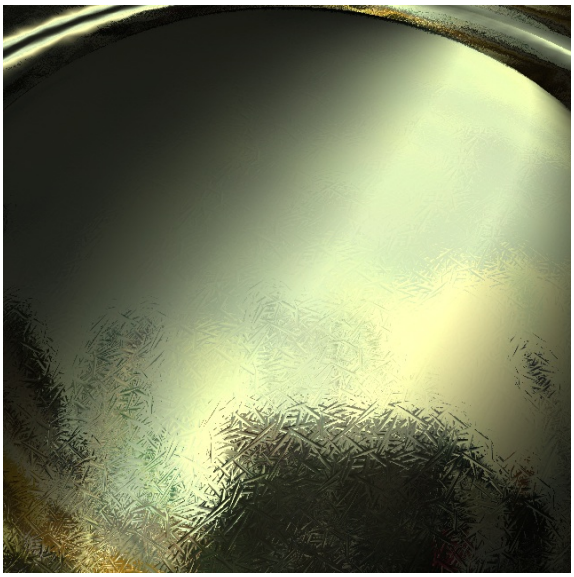
(c) Normal mapping

(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.

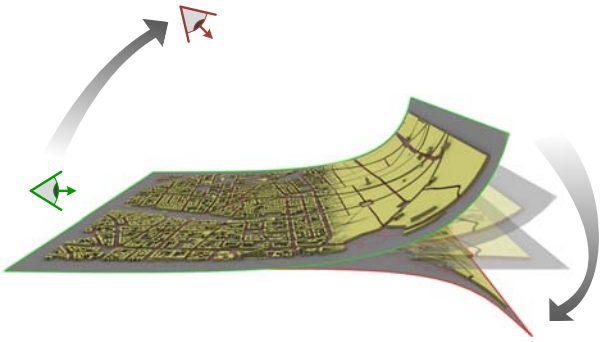


Figure 1: Conceptional sketch of the interpolation of global deformations and different geometric representations based on the viewing angle of the virtual camera.

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).

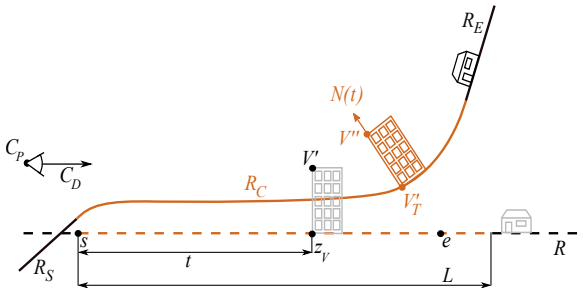


Figure 3: The reference plane R is separated by the parameter s and e into three sections: R_S, R_S and R_C. Based on the depth $z_{V'}$ of the vertex V along the camera direction C_D, the vertex is deformed onto one of the sections.

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.

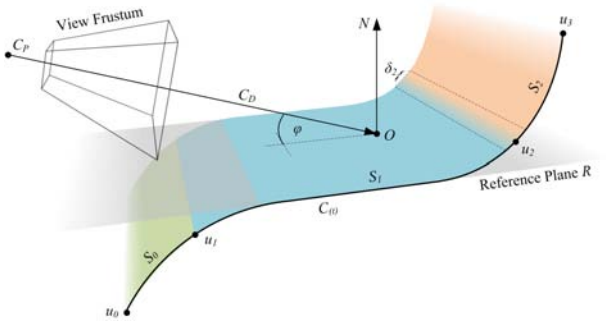


Figure 4: Exemplary parameterization of a deformation curve preset using four tag points (u_i).

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).

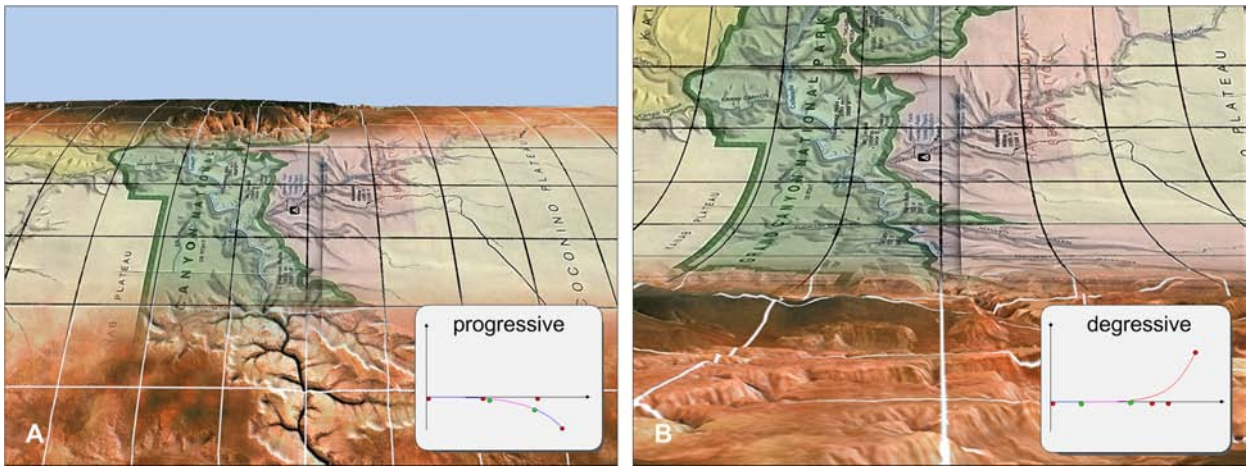


Figure 5: Styling section of a deformation curve with different models of the grand canyon. The inset shows the associated tag point and sections of the curve: The control points are depicted in red and the tag points are depicted green. The grid overlay was added to illustrate the deformation.

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 $\left[a_{P_S}, b_{P_S}\right]$ 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-

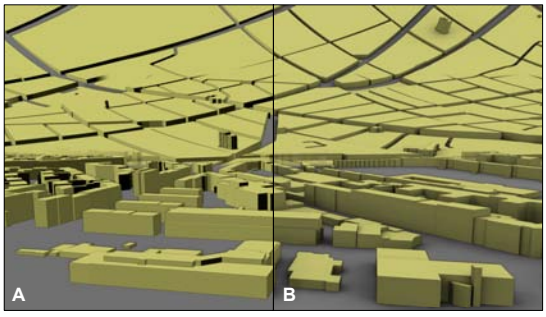


Figure 6: Comparison of applied stylization techniques for generalized virtual city models. A: Directional lighting and edge-enhancement. B: Precomputed ambient occlusion and edge-enhancement.

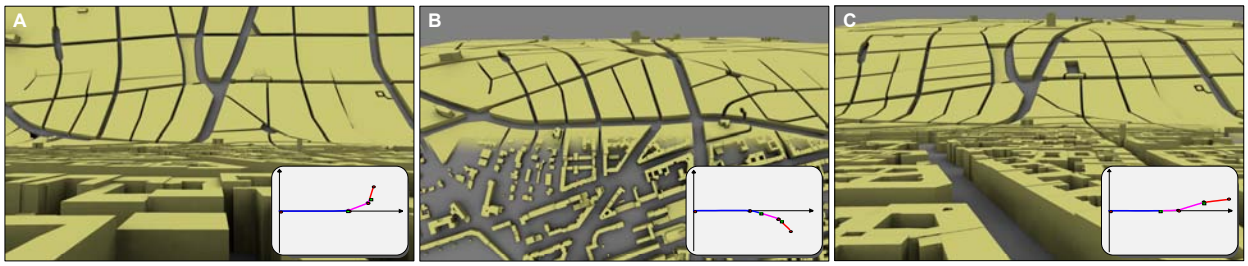


Figure 7: Exemplary visualization using B-Spline curves with six control points to enable hard transitions between three planar regions.

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).

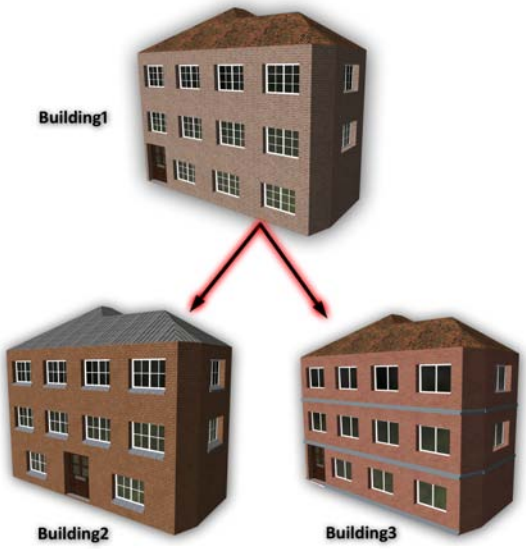


Figure 2: Building2 inherits from Building1, specifying how windowsills should be added. Building3 also inherits from Building1, adding a ledge to each floor. Code listings are provided in the Appendix.

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,

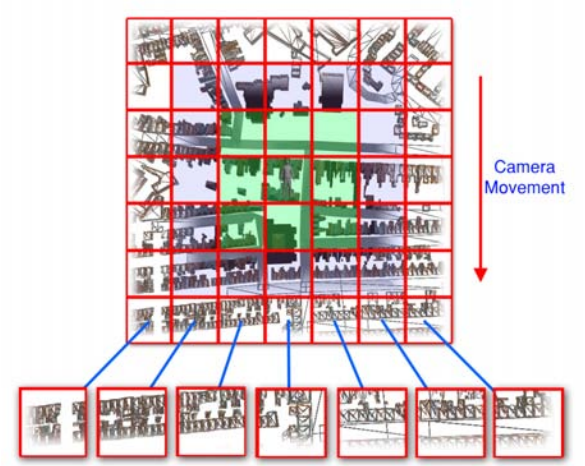


Figure 4: As the camera moves towards the geometry, new pages must be loaded. The pages are organised into a queue and processed in order of their distance to the camera. The buildings within each page should be shared among parallel executing threads.

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.

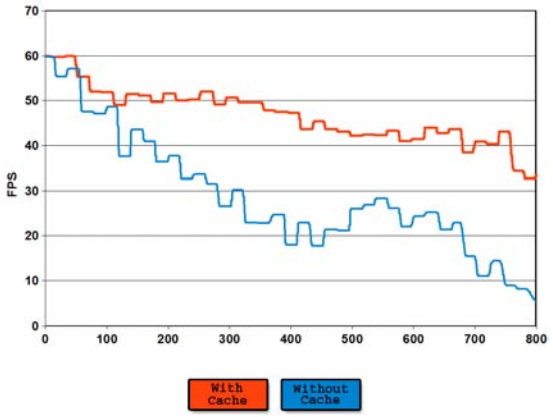
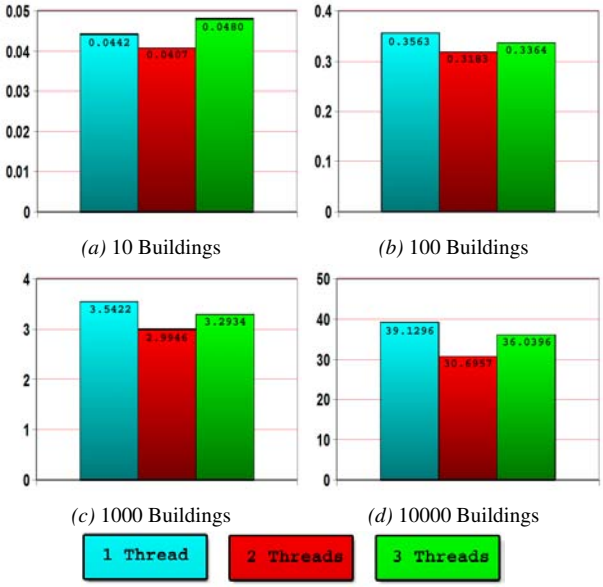


Figure 5: A comparison of results with and without the cache described in Section 5.1. The system with the cache has a much higher frame rate and less jerky movements of the camera. There was an average of 1,922 buildings in the scene at any time with 2,038 buildings created and destroyed over the distance.



(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.