ISSN 1213-6972 *Volume 14, Number 1-3, 2006*

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, human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

Journal of WSCG

Editor-in-Chief: Vaclav Skala
University of West Bohemia, Univerzitni 8, Box 314
306 14 Plzen
Czech Republic
skala@kiv.zcu.cz
Managing Editor: Vaclav Skala

Author Service Department & Distribution:
Vaclav Skala UNION Agency
Na Mazinach 9
322 00 Plzen
Czech Republic

Hardcopy: ***ISSN 1213 – 6972*** ***ISBN 80-86943-09-7***
CD ROM: ***ISSN 1213 – 6980***
On-line: ***ISSN 1213 – 6964***

WSCG 2006

International Programme Committee

WSCG 2006 Board of Reviewers

Rose,D. (Germany)
Rossignac,J. (United States)
Rudomin,I. (Mexico)
Sakas,G. (Germany)
Sanna,A. (Italy)
Sbert,M. (Spain)
Scateni,R. (Italy)
Segura,R. (Spain)
Shah,M. (United States)
Shamir,A. (Israel)
Schafhitzel,T. (Germany)
Schaller,N. (United States)
Scherzer,D. (Austria)
Schilling,A. (Germany)
Schneider,B. (United States)
Schneider,J. (Germany)
Scholz,V. (Germany)
Schumann,H. (Germany)
Sips,M. (Germany)
Sitte,R. (Australia)
Slusallek,P. (Germany)
Snoeyink,J. (United States)
Snoeyink,J. (United States)
Sochor,J. (Czech Republic)
Sojka,E. (Czech Republic)
Solis,A. (Mexico)
Sondershaus,R. (Germany)
Sporka,A. (Czech Republic)
Stephane,R. (France)
Stich,T. (Germany)
Strengert,M. (Germany)
Stroud,I. (Switzerland)
Stylianou,G. (Cyprus)
Sumanta,P. (United States)
Szekely,G. (Switzerland)
Szirmay-Kalos,L. (Hungary)
Tang,W. (United Kingdom)
Taubin,G. (United States)
Teschner,M. (Germany)
Theußl,T. (Austria)
Torres,J. (Spain)
Ulbricht,C. (Austria)
Van Laerhoven,T. (Belgium)
Vanecek,P. (Czech Republic)
Velho,L. (Brazil)
Veltkamp,R. (Netherlands)
Vergeest,J. (Netherlands)
Viola,I. (Austria)
VOLLRATH,J. (Germany)
Vuorimaa,P. (Finland)
Wan,T. (United Kingdom)
Weidlich,A. (Austria)
Weiskopf,D. (Canada)
Westermann,R. (Germany)
Wood,J. (United Kingdom)
Wu,S. (Brazil)
Wuethrich,C. (Germany)
Yilmaz,T. (Turkey)
Zach,C. (Austria)
Zachmann,G. (Germany)
Zalik,B. (Slovenia)
Zambal,S. (Austria)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)

Journal of WSCG

Vol.14, No.1-3

ISSN 1213-6972

Contents

(Additional files available on CD ROM version, only)

GPU-based Appearance Preserving Trimmed NURBS Rendering

Michael Guthe · Ákos Balázs · Reinhard Klein

Universität Bonn
Institute of Computer Science II
Computer Graphics
Römerstraße 164
53117 Bonn, Germany

guthe@cs.uni-bonn.de · edhellon@cs.uni-bonn.de · rk@cs.uni-bonn.de

ABSTRACT

Trimmed NURBS are the standard surface representation used in CAD/CAM systems and accurate visualization of trimmed NURBS models at interactive frame rates is of great interest for industry. To support modification and/or animation of such surfaces, a GPU-based trimming and tessellation algorithm has been developed recently. First, the NURBS is approximated with a bi-cubic hierarchy of Bézier patches on the CPU and then these are tessellated on the GPU. Since this approach only took the geometric error of an approximation into account, the various illumination artifacts introduced by the chosen bi-cubic approximation and the subsequent tessellation were neglected. Although this problem could be solved partially by calculating exact per-pixel normals on the GPU, the shading error introduced due to the bi-cubic approximation would remain. Furthermore, the long fragment shader required for per-pixel normals would lead to unacceptably low performance.

In this paper we present a novel bi-cubic approximation algorithm that takes the normal approximation error into account. In addition, we also define a new error measure to calculate the required grid resolution for the bi-linear approximation. In combination, this allows GPU-based NURBS tessellation with guaranteed visual fidelity. Our new method is also capable of high quality visualization of further attributes like curvature, temperature, etc. on surfaces with little or no modification.

Keywords GPU-based algorithms, NURBS tessellation, appearance preservation

1 INTRODUCTION

CAD/CAM systems used in industry for the design of models for prototyping and production are usually based on trimmed NURBS surfaces, since they have the ability to describe almost every shape conveniently. Additionally, the NURBS representation is also used more and more frequently to generate animations in movies or even for computer games.

Especially in CAD, but also in the growing field of virtual prototyping the accurate, real-time visualization of these NURBS models together with additional information – like reflection lines visualizing the quality of the model – becomes more and more important. However, recently developed techniques for real-time trimming and tessellation on commodity GPUs like [GBK05] do not take the appearance of the surface into account and only control the geometric error (see Figure 1). Such negligence of the normals required for correct shading can lead to severe visual artifacts.

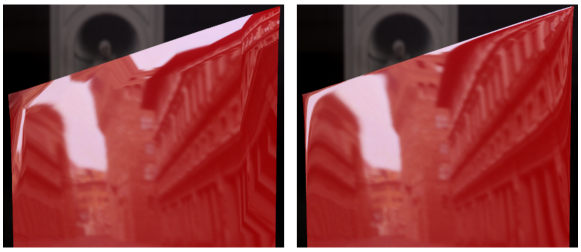


Figure 1: Difference between geometric (left) and appearance preserving (right) GPU-based tessellation.

Journal of WSCG, Vol.14, ISSN 1213-6972
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

In this paper we present a novel GPU-based rendering algorithm that also takes shading artifacts into account, but requires only a slight overhead compared to the original GPU-based tessellation algorithm [GBK05].
Even though the introduced error measures were originally developed for the approximation of surface normals, they are also well suited for high quality visualization of various other surface properties or attributes, such as curvature, temperature distribution, or basically any surface information that can be represented using scalar values or vectors.

2 PREVIOUS WORK

As our new method exploits ideas of appearance preserving tessellation and GPU-based tessellation, we give a short overview of both fields. Since higher order surfaces cannot be evaluated on GPUs efficiently, GPU-based methods are restricted to bicubic surfaces and, as a result, have to use degree reduction methods. Therefore, we also review prior work in this field. Finally, we give a brief survey of the state of the art in the field of surface property visualization.

2.1 Appearance Preserving Tessellation

An approach for view-dependent refinement of multiresolution meshes was developed by Klein et al. [KSS98] which could theoretically also be used for the tessellation of trimmed NURBS models. However, since their error measure is highly dependent on the position of the highlight and derivatives are calculated in screen space, the exact position and orientation of the surface on the screen to be known. This makes a complete retessellation of the model necessary in each frame. Since only a small portion of the surfaces can be retessellated on the CPU per frame, this approach was modified by Guthe et al. [GBK04] to become view-independent. However, this method still suffers from the high latency and inflexibility of CPU-based tessellation.

2.2 GPU-based Tessellation

Abi-Ezzi et al. [AES94] and Bóo et al. [BAD$^+$01] proposed an additional adaptive tessellation unit at the front of the rendering pipeline for NURBS and subdivision surfaces respectively. However, neither of these were built into commodity graphics hardware. Bolz and Schröder [BS03] developed an algorithm to evaluate Catmull-Clark subdivision surfaces on programmable graphics hardware. After the transmission of the tessellation textures to the GPU, only control points instead of triangles need to be send and thus the fragment shader can be saturated with marginal bus bandwidth consumption. With different tessellation textures this approach can also be used for bicubic B-Spline surfaces since they are equivalent to this subdivision scheme on a regular quad mesh. The algorithm generates an adaptive tessellation on a per-patch basis, which is rendered into an offscreen buffer – a so called pixel buffer or p-buffer – and then used as input for a second rendering pass. This method can achieve up to 30 million vertices per second on recent GPUs, but trimming of the surfaces is not possible. Based on this work, Kanai and Yasui [KY04] developed an algorithm to calculate accurate per-pixel normals on a tessellated subdivision surface. Although the produced images are very convincing, it is too slow for real time rendering at reasonable resolutions.
Recently a GPU-based trimming and tessellation algorithm for NURBS [GBK05] was developed. This method however, only takes the geometric error and not the shading error introduced due to incorrectly interpolated normals into account, which can lead to visual artifacts.

2.3 Degree Reduction

The idea of approximating high degree Bézier curves using degree reduction already came up more than 30 years ago [For72]. As shown by Park and Choi [PC95], the error can be reduced drastically by subdividing the curve before degree reduction. With a standard degree reduction algorithms, the degree of continuity between the composite curves cannot be controlled directly. Either, the continuity is preserved up to the maximum possible for the current curve degree (e.g. [For72]), or completely lost (e.g. [Eck93]). Therefore, Zheng and Wang [ZW03] developed a method to explicitly control the continuity classes of the curve at its endpoints. In [GBK05] a degree reduction method to preserve geometric continuity only has been presented. In contrast to all existing algorithms, which only consider the geometric error introduced by the degree reduction, the error measure proposed in this paper also takes the introduced shading error into account.

2.4 Surface Property Visualization

The rendering of surface properties is an important topic for surface interrogation and scientific visualization. Hagen et al. [HHS$^+$92] give an overview of different surface interrogation methods, like orthonomics, reflection lines and focal surfaces. In the context of our work we only concentrate on reflection lines, since they can be visualized on the surface. In addition to these properties, the visualization of the curvature and curvature regions [EC93a, EC93b] also delivers valuable information for surface design. For visualization so called property surfaces are generated in this approach. Since the calculation of these property surfaces is often computationally expensive, this method is not suited for complex or dynamic models.

3 GPU-BASED TESSELLATION

The overall workflow of the GPU-based trimming and tessellation algorithm [GBK05] is shown in Figure 2. First, the trimming curves are sampled with sufficient accuracy and evaluated on the GPU (2). Then the resulting polygons are rendered into a texture of appropriate size using a p-buffer (3). In the second rendering pass, the patch is sampled using a regular grid of sufficient resolution, such that a given geometric screen space error is guaranteed. For this purpose, predefined grids of different resolutions are stored on the graphics card in advance. At runtime, the grid index is calculated on the CPU and then sent to the GPU. Then the patch is evaluated at all grid vertices on the GPU (6). For the trimming, the trim-texture is simply bound and all pixels outside the trimming region are removed in the fragment stage by a lookup into this trim-texture (7).

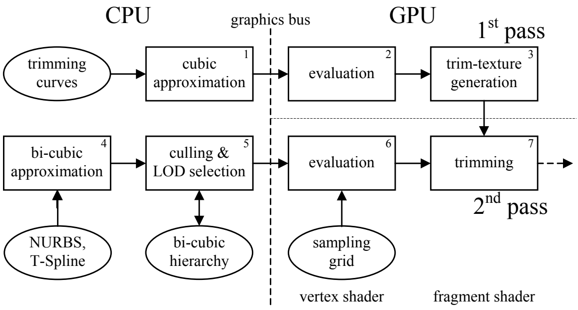


Figure 2: Main workflow of the GPU-based trimming and tessellation algorithm [GBK05].

As data dependent loops are only supported by very recent GPUs, a conversion from NURBS or T-Spline to piecewise rational Bézier representation is necessary, since the current knot spans, needed to calculate the sample points, differ. For cards not having texture access in the vertex shader, the amount of input data for a vertex program is limited to 16 vertex attributes and 8 program matrices and thus only low degree Bézier patches can be evaluated. To work with any graphics card supporting at least vertex shader 1.0, only 12 temporary registers can be used, which limits the maximum degree to bi-cubic. Therefore, the overall algorithm first approximates each NURBS or T-Spline surface and its trimming curves with a coarse hierarchy of rational bi-cubic Bézier patches, or cubic rational Bézier curves respectively, on the CPU (1+4). During rendering this hierarchy is traversed and patches with sufficient accuracy are selected to guarantee a given geometric screen space error (5). If the traversal reaches a leaf node, additional bi-cubic patches are generated. Then the control points of each patch are sent to the GPU before selecting a grid of appropriate resolution for evaluation.

An appearance – i.e. normal – preserving tessellation, based on this method, needs to preserve the normal in both approximation steps of the surface, namely the bi-cubic approximation on the CPU and the tessellation of the bi-cubic patch on the GPU. The remaining part of the algorithm however does not need to be changed. Therefore, the following two Sections describe only the modifications of the GPU-based NURBS rendering method necessary to preserve the appearance of the surfaces.

4 NORMAL PRESERVATION

When a Bézier surface S is approximated with a surface $\tilde{S}$, the visual approximation error on each point of the approximating surface $\tilde{S}(p)$, with the parameter value $p = (u, v)$, is the distance to the closest point on the original surface with the same color after shading, i.e. with the same normal $n = (n_x \ \ n_y \ \ n_z)^T$ when fragment based shading (e.g. Blinn-Phong or environment mapping) is used. This leads to the problem of finding a point $S(q)$ in the vicinity of $\tilde{S}(p)$, with $n(q) = \tilde{n}(p)$. Since we assume that S is smooth we can use a Taylor expansion of n around the parameter value p:

$$n(p+\Delta p) = n(p) + \begin{pmatrix} \frac{\partial n_x(p)}{\partial u} & \frac{\partial n_x(p)}{\partial v} \\ \frac{\partial n_y(p)}{\partial u} & \frac{\partial n_y(p)}{\partial v} \\ \frac{\partial n_z(p)}{\partial u} & \frac{\partial n_z(p)}{\partial v} \end{pmatrix} \Delta p + O(\|\Delta p\|^2)$$

Assuming $\|\Delta p\|$ to be small, a singular value decomposition could be used to find the smallest Δp and then $\|S(p + \Delta p) - \tilde{S}(p)\|$ is an upper bound for the visual error in object space. However, as shown in [GBK04], it is much more efficient to interpret the visual approximation error ε as the orthogonal combination of the geometric distance ε_{geom} of the point $\tilde{S}(p)$ on the approximating surface to the point $S(p)$ on the original surface and the distance ε_{norm} of $S(p)$ to the closest point $S(p + \Delta p)$ with the same normal $n(p + \Delta p) = \tilde{n}(p)$. As shown in Figure 3, these two distances can be combined by:

$$\varepsilon(p)^2 = \varepsilon_{geom}(p)^2 + \varepsilon_{norm}(p)^2.$$

In this case we are able to exploit the fact that the estimation of the geometric approximation error remains the same as for the non-appearance preserving tessellation. Thus the shading error can be estimated without actually calculating the position of the closest correctly shaded point in Euclidean space, using the approximate error measures deduced in the following.

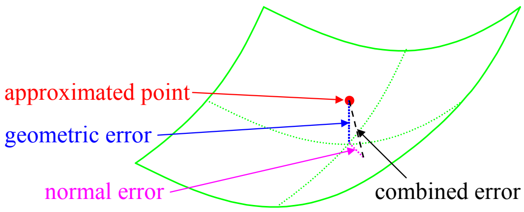


Figure 3: Combination of error measures.

4.1 Bi-cubic Approximation

When evaluating a Bézier surface, the normal is calculated as the cross-product of the first derivatives in u- and v-direction. This implies, that the normal on an approximating surface $\tilde{S}$ at parameter p equals that of the original surface S at parameter q, if

$$\frac{\partial \tilde{S}(p)}{\partial u} = \frac{\partial S(q)}{\partial u} \quad \text{and} \quad \frac{\partial \tilde{S}(p)}{\partial v} = \frac{\partial S(q)}{\partial v}.$$

Since the bi-cubic approximation of the arbitrary degree Bézier surface S with a bi-cubic Bézier surface $\tilde{S}$ is performed first in the u- and then in the v-direction, preserving the normal can be achieved by preserving the first derivative of each iso-parametric curve. The derivative approximation error ε_d when approximating a curve C with $\tilde{C}$ is then

$$\varepsilon_d(t) = \|C'(t) - \tilde{C}'(t)\|.$$

Since this error is defined in the space of the first derivative, it needs to be projected into object-space. This projection needs to map a distance δ_d in derivative space to a distance δ_o in object-space. Again we approximate the distances on the curves using a Taylor expansion around t:

$$\begin{aligned} \delta_o(t+\Delta t) &= \Delta t C'(t) + O(\|\Delta t\|^2) \\ \delta_d(t+\Delta t) &= \Delta t C''(t) + O(\|\Delta t\|^2) \end{aligned}$$

For a small Δt, we can approximate the projection with $\delta_o(t+\Delta t) \approx \Delta t C'(t)$ and $\delta_d(t+\Delta t) \approx \Delta t C''(t)$ such that the object-space derivative error $\varepsilon_{der}(t)$ between the two curves at parameter t is then

$$\varepsilon_{der}(t) \approx \left(\|C'(t) - \tilde{C}'(t)\| \frac{\|C'(t)\|}{\|C''(t)\|} \right).$$

The object-space derivative deviation error ε_{der} between the original and approximating curve is now defined as the maximum of $\varepsilon_{der}(t)$ along the curve:

$$\varepsilon_{der} \approx \sup_{0 \le t \le 1} \left(\|C'(t) - \tilde{C}'(t)\| \frac{\|C'(t)\|}{\|C''(t)\|} \right).$$

Arguing similarly to [GBK04], we can assume that the maximum derivative deviation error on a curve will probably occur at the point, where $\|C''(t)\|$ has its maximum and therefore the following approximation can be used without loss of visual fidelity:

$$\varepsilon_{der} \approx \sup_{0 \le t \le 1} \|C'(t) - \tilde{C}'(t)\| \frac{\sup_{0 \le t \le 1} \|C'(t)\|}{\sup_{0 \le t \le 1} \|C''(t)\|}.$$

4.2 Sampling

To generate less rendering primitives (e.g. for surfaces of revolution), the sampling resolution in u- and v-direction is separated as in [GBK05]. According to Filip et al. [FMM86], the error ε when approximating a C^2-continuous surface with a regular triangle mesh, where each pair of triangles spans the bilinear parameter space rectangle $D = [(u_i, v_j), (u_{i+1}, v_{j+1})]$ with the constant sizes $\Delta u = u_{i+1} - u_i$ and $\Delta v = v_{j+1} - v_j$ is bounded by

$$\varepsilon \le \frac{1}{8}(\Delta u^2 M_u + 2\Delta u \Delta v M_{uv} + \Delta v^2 M_v),$$

with

$$M_u = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial u^2} \right\|, \quad M_{uv} = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial u \partial v} \right\|, \quad \text{and } M_v = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial v^2} \right\|.$$

The sampling densities are then separated by exploiting the fact that $ab \le \frac{1}{2}(a^2 + b^2)$ and thus the approximation error is bound by

$$\varepsilon \le \frac{1}{8}\left(\Delta u^2 (M_u + M_{uv}) + \Delta v^2 (M_v + M_{uv}) \right),$$

which is a simple addition of the two approximation errors in u- and v-directions. This means, that ε is an upper bound for the approximation error, if the error in both directions is not greater than $\frac{\varepsilon}{2}$. This can further be simplified to calculating the piecewise linear approximation error of $n + m$ curves.
Following the estimations proposed in Section 4.1, we again assume that the maximum derivative error occurs at the point where $\|C''(t)\|$ has its maximum, which leads to the following approximate derivative deviation error:

$$\varepsilon_{der}(t) \approx \|C'(t) - \tilde{C}'(t)\| \frac{\sup_{0 \le t \le 1} \|C'(t)\|}{\sup_{0 \le t \le 1} \|C''(t)\|}.$$

Since $\varepsilon_{der}(t)$ is C^2 continuous, if C is C^3 continuous, which is the case for cubic Bézier curves, the theorem of Filip et al. [FMM86] gives an approximate upper bound ε_{der} of a piecewise linear approximation with a constant step size d of

$$\varepsilon_{der} \approx \frac{1}{8} d^2 \sup_{0 \le t \le 1} \|C'''(t)\| \frac{\sup_{0 \le t \le 1} \|C'(t)\|}{\sup_{0 \le t \le 1} \|C''(t)\|}.$$

The number of required samples n to achieve a maximum given deviation of ε is then:

$$n = \left\lceil \sqrt{\frac{\sqrt{E_{geom}^2 + E_{der}^2}}{8\varepsilon}} \right\rceil ,$$

with

$$E_{geom} = \sup_{0\le t\le 1} \|C''(t)\|$$
$$E_{der} = \sup_{0\le t\le 1} \|C'''(t)\| \frac{\sup_{0\le t\le 1} \|C'(t)\|}{\sup_{0\le t\le 1} \|C''(t)\|}.$$

$C'''(t)$ can be written as a rational Bézier curve with a degree nine nominator $\check{P}(t) = \sum_{i=0}^{9} \check{P}_i B_i^9(t)$ and a degree twelve denominator $\check{w}(t) = \sum_{i=0}^{12} \check{w}_i B_i^{12}(t)$. Since all w_i are positive by construction, all $\check{w}_i$ are also positive. Therefore, an upper bound of the norm of the third derivative is given by:

$$\sup_{0\le t\le 1} \|C'''(t)\| \le \frac{\max(\|\check{P}_0\|, \ldots, \|\check{P}_9\|)}{\min(\check{w}_0, \ldots, \check{w}_{12})}.$$

The upper bounds for $\|C''(t)\|$ and $\|C'(t)\|$ are calculated as in [GBK05], i.e. $\|C''(t)\|$ from a degree seven/nine and $\|C'(t)\|$ from a degree five/six rational polynomial curve. Since the calculation of these upper bounds is only required when extending the bi-cubic hierarchy, the additional computation time can be expected to be marginal for static models.

5 OTHER ATTRIBUTES

The appearance preserving error measure derived in Section 4 is not limited to normals – which are preserved when preserving the first derivatives – but can easily be extended to higher derivatives or arbitrary attributes. If $A(u, v)$ is a general attribute defined as a tensor product, we can again reduce the problem into a piecewise curve representation and project the approximation error from attribute- to object-space with

$$\varepsilon_A(t) = \|C_A(t) - \tilde{C}_A(t)\| \frac{\|C'(t)\|}{\|C'_A(t)\|}.$$

Starting from this definition, the approximation error required for the bi-cubic approximation and the regular tessellation can be derived using the same assumptions and estimations as in Section 4. For the bi-cubic approximation, we then have

$$\varepsilon_A \approx \sup_{0\le t\le 1} \|C_A(t) - \tilde{C}_A(t)\| \frac{\sup_{0\le t\le 1} \|C'(t)\|}{\sup_{0\le t\le 1} \|C'_A(t)\|},$$

and for the sampling resolution

$$\varepsilon_A \approx \frac{1}{8d^2} \sup_{0\le t\le 1} \|C''_A(t)\| \frac{\sup_{0\le t\le 1} \|C'(t)\|}{\sup_{0\le t\le 1} \|C'_A(t)\|}.$$

Finally, the approximation errors of all attributes are combined with the geometric approximation error as an orthogonal combination of partial errors.

6 RESULTS

To evaluate the efficiency of our method, we first compare its performance with the previous GPU-based tessellation method, that only guarantees a geometric error. Then we examine the image quality improvements provided by our new method and finally we test its applicability in the field of surface property visualization, especially in comparison with the previous method.

6.1 Performance

All benchmarks were performed on an Athlon 64 3200+ with 1.5 GByte memory and a GeForce 7800 GTX at a resolution of 1280×1024 (unless noted otherwise) with 0.5 pixel screen space error.

First, we compare the tessellation performance of our method with the performance of the previous GPU-based algorithm using a single bi-cubic trimmed and untrimmed patch (see Figure 4). To analyze the tessellation performance we render this patches at different screen-sizes, where a larger screen-size implies a lower object-space error and a higher sampling rate.

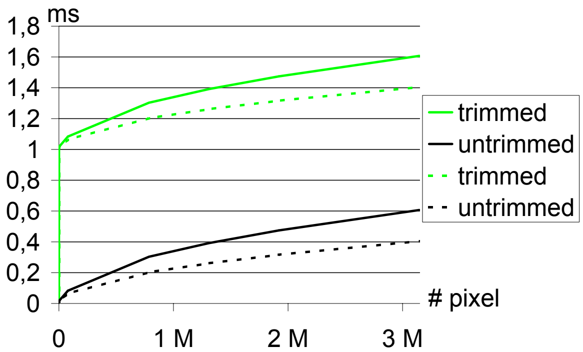


Figure 4: Tessellation performance in dependance of screen size for geometric (dashed) and appearance preserving GPU-based tessellation.

As shown by these graphs, the number of additional vertices and thus the additional rendering time for the untrimmed surface is approximately 50%. However, when the surface is very small on screen (a few pixel), the additional number of vertices and the performance loss is approaching zero. It can also be observed, that the trimming overhead remains constant, since the trimming itself does not alter the appearance of the surface and remained unchanged.

As second example, we compare the performance of the bi-cubic approximation for surfaces of different degrees with that of the original GPU-based tessellation.

 ISBN 1213-6972 ISBN 80-86943-09-7

In Figure 5 the performance of both methods is shown for a single animated trimmed NURBS surface with 100 control points and degrees of 3×3, 5×5, and 7×7 respectively.

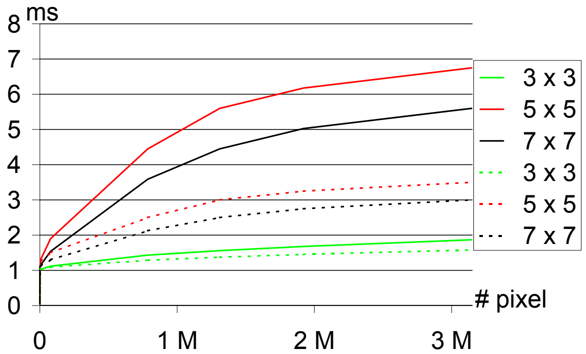


Figure 5: Total rendering performance of a single animated trimmed NURBS surface with 100 control points and of different degrees using geometric (dashed) and appearance preserving GPU-based tessellation.

For large surfaces of higher degree, where a bi-cubic approximation is required, the total rendering time increases by up to 93%. This performance drop is mainly due to the computationally more expensive error measure for the bi-cubic approximation, as the percentage of additional bi-cubic patches and rendered vertices is significantly lower than this. Since the bi-cubic approximation error measure needs to be calculated only when a surface is modified, the impact on static models will be significantly lower.

To evaluate the performance on more complex static models, resembling a real application setting, we render the industrial CAD models shown in the Figures 6-8.

Figure 6: Mini model: 629 trimmed surfaces.

Detailed statistics on the number of NURBS and underlying Bézier surfaces, as well as the number of non-trivially trimmed NURBS surfaces, of these models are given in Table 1.

Table 2 compares the average number of rendered bi-cubic Bézier patches, the average number of generated

Figure 7: Golf model: 8,138 trimmed surfaces.

Figure 8: C-Class model: 67,571 trimmed surfaces.

	Mini	Golf	C-Class
NURBS surfaces	629	8,138	67,571
non-trivially trimmed	203	1,486	35,230
Bézier patches	25,648	17,936	396,535

Table 1: Details of the models used for evaluation.

vertices and the frame-rate of the unmodified GPU-based trimming and tessellation algorithm [GBK05] with the appearance preserving method presented in this paper.

	Mini	Golf	C-Class
geometric only approximation			
bi-cubic patches	11,683	8,008	105,442
vertices	210,529	239,221	2,216,352
frame-rate	12.8 fps	9.1 fps	1.3 fps
appearance preserving approximation			
bi-cubic patches	11,685	8,017	159,176
vertices	283,333	280,400	2,538,128
frame-rate	11.0 fps	7.9 fps	1.2 fps

Table 2: Performance comparison between geometric and appearance preserving approximation.

Even though the number of bi-cubic patches has increased by 21% to 51% and the number of vertices increased by 14% to 39%, the frame-rate difference is only between 8% and 14%. The very loose coupling between the number of vertices and the rendering performance is mainly due to the massively parallel architecture of modern GPUs. With n parallel vertex units, the evaluation and transformation time t of each bicubic Bézier patch with v vertices is

$$t = c\left\lceil\frac{v}{n}\right\rceil,$$

where c depends on the GPU performance. Note, that for these models, the average number of vertices per Bézier patch (16 to 30) is in the same order of magnitude as the number of parallel vertex units in current GPUs, even for the appearance preserving tessellation. In addition to this, each Bézier patch requires a constant time for initialization of the vertex array, uploading of the control points for evaluation, and setting the domain interval for trimming, regardless of the size of the regular grid used for evaluation. Furthermore, the time required for trimming remains constant as well.

6.2 Image Quality

In order to compare with the previous, purely geometric approach, we perform a pixel by pixel comparison of the interpolated normals with the real normals from the NURBS model obtained via sub-pixel subdivision. The visual difference between the real and interpolated normals (shown in Figure 9) can be extracted using simple image processing. To estimate the normal deviation in screen space, the surrounding pixels are used to calculate the normal derivatives. Note, that this is only correct on a closed surface but not along contours.

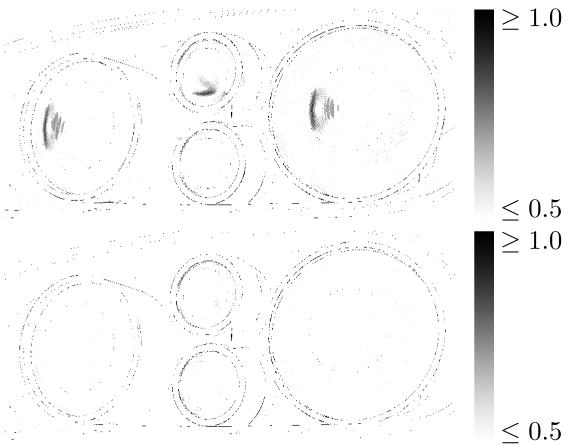


Figure 9: Normal deviation error in pixels for a closeup of the Golf model using GPU-based tessellation without (top) and with normal preserving error measure (bottom).

It is clearly visible that the normal approximation is much better when using the appearance preserving tessellation. Note, that the remaining pixels along contours, where the normal error exceeds the 0.5 pixel threshold are – as already mentioned – due to the normal undersampling in the image processing step and not a shortcoming of the appearance preserving tessellation algorithm.

In addition to the visual comparison, Table 3 compares the average normal deviation of our approach with the previous GPU-based tessellation algorithm that controls the geometric error only.

	angle	pixel	exceeded
geometric only	$0.930°$	0.117	2.409%
normal preserving	$0.752°$	0.074	1.238%

Table 3: Average normal approximation error per foreground pixel and percentage of foreground pixels where desired error is exceeded.

Here again, the remaining pixels where the screen space error threshold is exceeded are located along contours and are therefore the results of aliasing artifacts and not due to incorrect normals.

6.3 Surface Properties

The main goal of surface property visualization in industry, especially in design, is to ensure the continuity of reflections on the surface. For this purpose, so-called reflection lines are mainly used. Figure 10 compares the reflections lines rendered with a grid environment on a closeup of the Golf fender using geometric and appearance preserving tessellation.

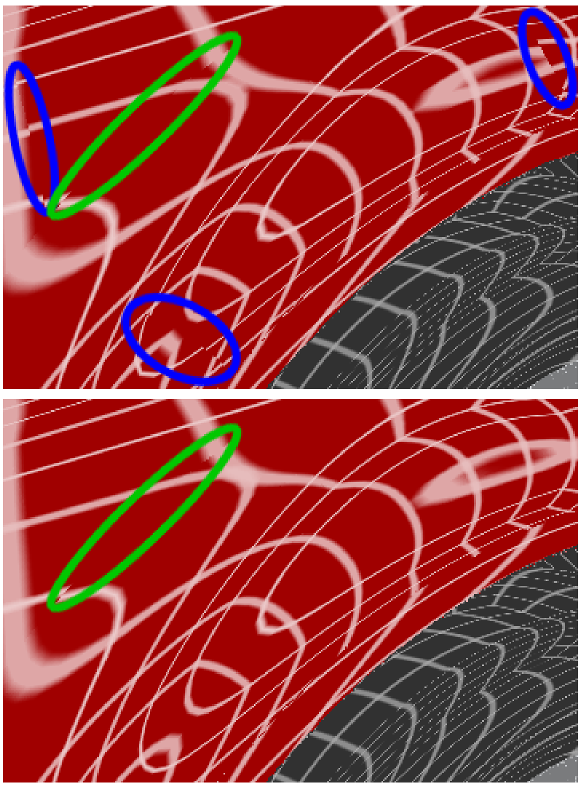


Figure 10: Reflection lines using geometric (top) and appearance preserving (bottom) tessellation. In the top image, the real discontinuity (green) is indistinguishable from tessellation related (blue).

 ISBN 1213-6972 ISBN 80-86943-09-7

To identify discontinuities in the shading, which occur at ridges or ravines of the model, the tessellation needs to produce meshes with correct normal interpolation. Even a slight normal deviation of a few degree can lead to visual artifacts that are indistinguishable from real surface discontinuities. Using the appearance preserving bi-cubic approximation and tessellation presented in this paper, the normals are correct within a given screen space error and thus shading discontinuities only occur when they are present in the model.

7 CONCLUSION

In this work, we presented a novel method for GPU-based appearance preserving tessellation of NURBS surfaces. We demonstrated the problems of the previous algorithm in dealing with various illumination/shading artifacts introduced by the bi-cubic approximation and the following tessellation. We also demonstrated that our algorithm only requires a relatively low number of additional bi-cubic patches and vertices to produce accurate interpolated normals. It achieves almost the same performance as the original method, but nevertheless provides a much higher visual fidelity. Our new method also has the capability to visualize surface properties such as degree of continuity or discontinuities using reflection lines. Due to the real-time trimming and tessellation on the GPU, it is also suitable for the visualization of deformable models and to have immediate feedback during the design and virtual prototyping process.

8 ACKNOWLEDGEMENTS

We would like to thank SGI, DaimlerChrysler AG, and Volkswagen AG for providing for the trimmed NURBS models used in this paper. This work was partially funded by the European Union under the project of “Real Reflect” (IST-2001-34744).

References

[AES94] S. S. Abi-Ezzi and S. Subramanian. Fast dynamic tessellation of trimmed NURBS surfaces. *Computer Graphics Forum*, No.13(3), pp.107–126, 1994.

[BAD+01] M. Bóo, M. Amor, M. Doggett, J. Hirche, and W. Straßer. Hardware support for adaptive subdivision surface rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp.33–40, 2001.

[BS03] J. Bolz and P. Schröder. Evaluation of subdivision surfaces on programmable graphics hardware, 2003.

[EC93a] G. Elber and E. Cohen. Hybrid symbolic and numeric operators as tools for analysis of freeform surfaces. In *Working Conference on Geometric Modeling in Computer Graphics*, pp.275–286, 1993.

[EC93b] G. Elber and E. Cohen. Second-order surface analysis using hybrid symbolic and numeric operators. *ACM Transactions on Graphics*, No.12(2), pp.160–178, 1993.

[Eck93] M. Eck. Degree reduction of Bézier curves. *Computer Aided Geometric Design*, No.10(3-4), pp.237–252, 1993.

[FMM86] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design*, No.3(4), pp.295–311, 1986.

[For72] A. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer Journal*, No.15(1), pp.71–79, 1972.

[GBK04] M. Guthe, Á. Balázs, and R. Klein. Interactive High Quality Trimmed NURBS Visualization Using Appearance Preserving Tessellation. In *Data Visualization 2004 (Proceedings of TCVG Symposium on Visualization)*, pp.211–220 + 348. EUROGRAPHICS - IEEE, May 2004.

[GBK05] M. Guthe, Á. Balázs, and R. Klein. GPU-based trimming and tessellation of NURBS and T-Spline surfaces. *ACM Transactions on Graphics*, No.24(3), pp.1016–1023, 2005.

[HHS+92] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wördenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. In *IEEE Visualization and Computer Graphics*, pp.53–60, 1992.

[KSS98] R. Klein, A. Schilling, and W. Straßer. Illumination dependent refinement of multiresolution meshes. In *Proceedings of Computer Graphics International (CGI ’98)*, pp.680–687, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[KY04] T. Kanai and Y. Yasui. Per-pixel evaluation of parametric surfaces on GPU. In *ACM Workshop on General Purpose Computing Using Graphics Processors (also at SIGGRAPH 2004 poster session)*, August 2004.

[PC95] Y. Park and U J. Choi. Degree reduction of Bézier curves and its error analysis. *J. Austral. Math. Soc. Ser. B*, No.36, pp.399–413, 1995.

[ZW03] J. Zheng and G. Wang. Perturbing Bézier coefficients for best constrained degree reduction in the L_2-norm. *Graphical Models*, No.65, pp.351–368, 2003.

Optimizing GPU Volume Rendering

Daniel Ruijters
Philips Medical Systems
Veenpluis 6
5680DA Best, the Netherlands
danny.ruijters@philips.com

Anna Vilanova
Technische Universiteit Eindhoven
Den Dolech 2
5600MB Eindhoven, the Netherlands
a.vilanova@tue.nl

ABSTRACT

Volume Rendering methods employing the GPU capabilities offer high performance on off-the-shelf hardware. In this article, we discuss the various bottlenecks found in the graphics hardware when performing GPU-based Volume Rendering. The specific properties of each bottleneck and the trade-offs between them are described. Further we present a novel strategy to balance the load on the identified bottlenecks, without compromising the image quality. Our strategy introduces a two-staged space-skipping, whereby the first stage applies bricking on a semi-regular grid, and the second stage uses octrees to reach a finer granularity. Additionally we apply early ray termination to the bricks. We demonstrate how the two stages address the individual bottlenecks, and how they can be tuned for a specific hardware pipeline. The described method takes into account that the rendered volume may exceed the available texture memory. Our approach further allows fast run-time changes of the transfer function.

Keywords

Volume Visualization, Direct Volume Rendering, Texture Slicing, Hierarchical Rendering, GPU.

1. INTRODUCTION

New developments in medical imaging modalities, numerical simulations, geological measurements, etc. lead to ever increasing sizes in volumetric data. The ability to visualize and manipulate the 3D data interactively is of great importance in the analysis and interpretation of the data. The interactive visualization of such data is a challenge, since the frame rate is heavily depending on the amount of data to be visualized. Inherently, the demand for faster visualization methods is always existing, in spite of hardware innovations.

An established method for interactive volume rendering on consumer hardware is GPU-based texture slicing [Ake93, CCF94, CN93, EE02, EKE01, MGS02, RGW+03, KW03]. Although this approach performs very well compared to CPU-based algorithms, since it benefits from the parallelism available in the GPU pipeline, it can be accelerated significantly by taking into account the various bottlenecks that are encountered in the graphics hardware. Every individual bottleneck has a different optimal data chunk size and data throughput. In this article, we present a novel approach to accelerate GPU-based volume rendering that allows to tailor and balance the load on the individual bottlenecks to reach an optimal exploitation of the graphics hardware power.

In section 2, we present an overview of related work. Section 3 discusses the main bottlenecks that come into play when performing GPU-based volume rendering. Then an outline of the proposed approach is drawn in section 4. Sections 5, 6 and 7 deal with the details of our approach. In section 8, the results are presented and discussed, and in section 9 we summarize our conclusions.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

2. RELATED WORK

The first rendering methods using the 3D texture capabilities of the graphics hardware were proposed by Cullip and Neumann [CN93], Akeley [Ake93] and Cabral et al. [CCF94]. Essentially these techniques consist of drawing polygons, which slice the volume in a back to front order. The data set is mapped as texture information on the polygons using tri-linear interpolation. The successive polygons are blended into the existing image.

Bricking is a technique to divide the volume data set into chunks, called bricks [Eck98, WWE04]. It can be employed to deal with data sets exceeding the available texture memory. The bricks have then a size that is equal to or smaller than the size of the texture memory, and are loaded sequentially from main memory into the texture memory while rendering. However, this leads to significantly lower frame rates, since the bus architecture, connecting the graphics hardware to the main memory and CPU, proves to be a major bottleneck. Tong et al. [TWTT99] propose a bricking technique that allows skipping empty regions. Their method, however, requires new textures to be generated for every change of the transfer function, which is time consuming for very large data sets.

Texture compression can help to fit the entire volume in the main memory, and to alleviate the bus bottleneck. However, all presently available compression methods supported by graphics hardware (S3TC, FXT1, DXT1, VTC, etc) are limited to lossy 8-bit RGB(α) compression, which make them unsuitable for the compression of the (often 12- or 16-bit) scalar values found in medical data, and therefore we do not use them. Further, Meissner et al. [MGS02] show that the lossy compression algorithms severely reduce the image quality. Wavelet compression, as proposed by Guthe et al. [GWGS02] is a promising technique, but there, not all parts of the volume are rendered at the highest resolution.

Not rendering all parts of the volume in the highest resolution possible is a way to reach higher frame rates, as demonstrated by LaMar et al. [LHJ99], Weiler et al. [WWH+00], Boada et al. [BNS01] and Guthe et al. [GWGS02]. This is particularly suited to increase the render speed for perspective projections in a small view port, focusing on a detail of the volume. However, orthogonal projections of the entire volume in high resolution view ports, as is common in medical applications, can only profit from this technique at the cost of the image quality.

Space-skipping and space-leaping are techniques to accelerate volume rendering, that origin from ray-casting methods, see e.g. Levoy [Lev90], Zuiderveld et al. [ZKV92] and Yagel and Shi [YS93]. It is based on skipping empty parts of the volume. The idea of space-skipping can be applied to texture-mapping volume rendering as has been shown by Westermann and Sevenich [WS01].

Octree is an established multi-level data structure when dealing with voxel volumes, which has been used in numerous different applications. E.g. Srinivasan et al. [SFH97] apply an octree structure in volume rendering. Orchard and Möller [OM01] demonstrated the benefits of using adjacency information in splatting volume rendering.

Parker et al. have combined bricking and multi-level data structures to accelerate CPU-based iso-surface ray-tracing of volume data sets on multi-processor platforms and clusters [PSL+99, DPH+03]. Grimm et al have applied a two-staged space skipping, based on bricking and octrees, combined with gradient caching, to CPU-based ray-casting [GBKG04].

Roettger et al. [RGW+03] describe a GPU-based pre-integrated texture-slicing including advanced lighting. The authors also describe a GPU-based ray-tracing approach with early ray termination. Krüger and Westermann [KW03] propose a method to accelerate volume rendering based on early ray termination and space-skipping in a GPU-based ray-casting approach. The space-skipping addresses the rasterization bottleneck, using a single octree level only.

We have combined some of the techniques cited above, to accelerate GPU volume rendering on a single workstation, using off-the-shelf hardware. Often we found that acceleration of volume rendering has been treated as a singular problem to solve. We rather focus on the individual bottlenecks that are encountered while performing volume rendering, and tailor the different techniques to address specifically those bottlenecks.

3. BOTTLENECKS

Figure 1 illustrates the graphics pipeline, employed for GPU-based volume rendering [Zel02]. Here we discuss the most important points in the pipeline that result in a bottleneck.

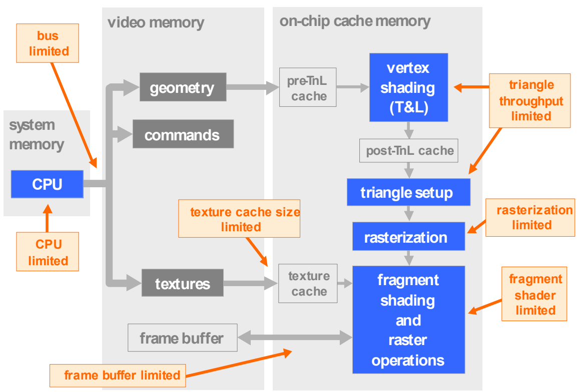
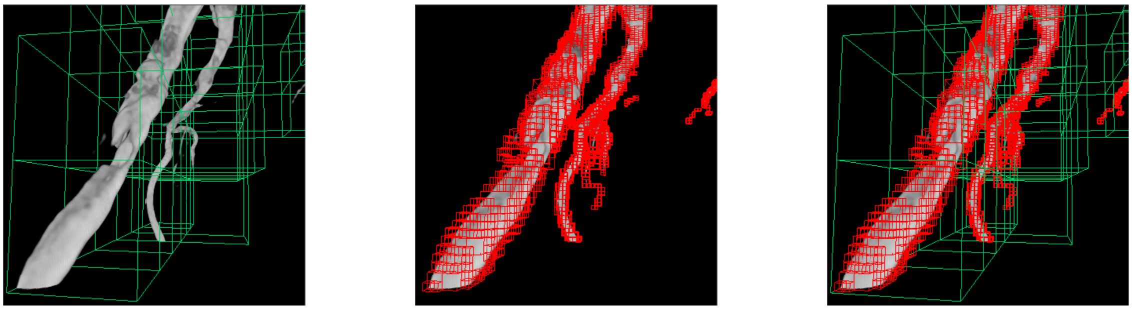


Figure 1: The graphics hardware pipeline and its bottlenecks [Zel02], light grey: memory units, dark grey: data structures, blue: processing units, red: bottlenecks.

The bus - The volume data has to be transferred over the bus from the system memory into the

 ISBN 1213-6972 ISBN 80-86943-09-7



(a) **(b)** **(c)**

Figure 2: The same volume fragment, rendered with (a) bricking cubes visible, (b) octree cubes visible (note the various cube sizes) and (c) both bricking and octree cubes visible

graphics card memory. Since this is the slowest part of the entire pipeline, these transfers have to be as few as possible.

Triangle throughput - The triangle throughput is mainly limited by the vertex shading and triangle setup phase. A straight forward implementation of texture-mapping volume rendering would involve only few triangles, but techniques for space-skipping may increase the amount of triangles considerably. If the triangle count becomes too high, this will become a limiting factor for the frame rate.

Rasterization - When performing volume rendering based on texture slicing, the vast majority of the pixels on the screen are accessed multiple times. Space-skipping techniques may be used to reduce the amount of pixels to be accessed, but this also increases the triangle count.

Texture cache size - Texture lookup is one of the more time consuming operations performed during the rasterization step. When the texture fits in the cache, these lookup operations will be faster.

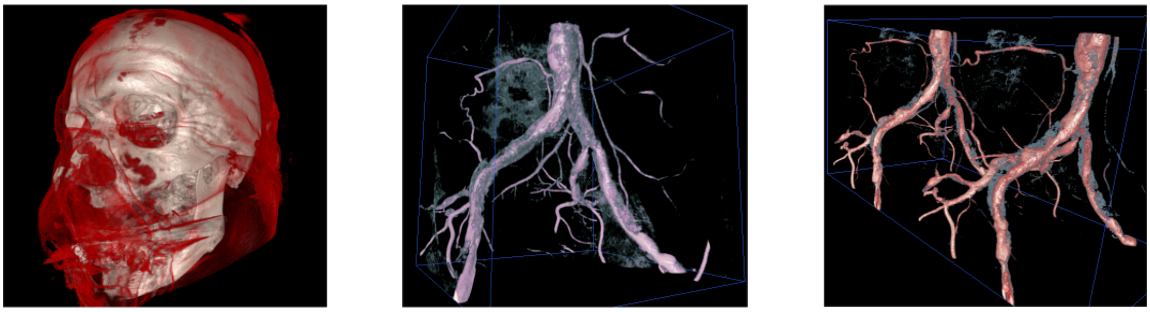
Fragment shader - Fragment shader programs impact the duration of the rasterization step. Simple fragment programs, such as applying a lookup table, generally do not limit the frame rate, however more complex operations, such as specular lighting [MGS02, RGW+03], multi-dimensional transfer functions [KKH01] or pre-integrated rendering [EE02, EKE01, RGW+03], can form a bottleneck. Especially fragment programs that perform multiple texture lookups (e.g. on-the-fly gradient calculation for specular lighting) are relatively slow.

4. OUR APPROACH

When performing volume rendering usually only a fraction of all voxels actually contribute to the final image, since a relatively small amount of voxels are of interest and another amount of them are occluded. In 3D medical data sets (obtained by e.g. ultrasound, CT, MR or rotational angiography [KodBA98, vdB03]) the anatomical structures of interest encapsulated in the data sets occupy only a part of the total data. Typically 5% to 40% of all voxels contain visible data, and even highly filled CT or MR data sets rarely exceed 55%. Especially vascular data sets can be marked as sparse data sets, since vessels, due to their tubular form, occupy only a small percentage of the volume (1% to 8%).

In this article, we seek to reach the maximum benefit in exploiting skipping void parts of the volume (space-skipping). The novelty we introduce lies in dividing the space-skipping in two stages; a course division using bricking (figure 2a) and a finer one using octrees (figure 2b). These steps are based on an analysis of the bottlenecks encountered in the graphics pipeline when performing texture-mapping volume rendering. The first stage, bricking, is chopping the volume in so called texture bricks. The bricks are loaded into the video memory, to serve as data for the volume rendering algorithm, which is executed by the GPU. The bricks address the bus- and texture cache size-bottleneck. To further alleviate the load on the fragment shaders, we additionally perform early ray termination to each brick. This benefits especially highly-filled data sets. The second stage is employing an octree within each brick. The octrees address the rasterization bottleneck. As we demonstrate, the two stages have to be balanced, because lifting one bottleneck may overload another bottleneck (e.g. rasterization bottleneck versus triangle throughput bottleneck).

The role of the transfer function in volume rendering is to map the scalar voxel information to optical properties (e.g. color and opacity) [KKH01]. The above described approach is implemented such that the flexibility to change the transfer function at run-time is preserved. This offers the possibility to focus on different scalar ranges in the volume, without lengthy calculations. To accomplish this, the unmodified scalar voxel values are stored in the brick textures, and a fragment shader program is used, to lookup the RGBα values post-interpolatively.



(a) (b) (c)

Figure 3: Test volumes: (a) 512^3 volume, used for testing early ray termination, (b) vascular 512^3 volume, (c) gigabyte volume of $642 \cdot 642 \cdot 1284$ voxels, generated by duplicating a large 3D-RA volume.

5. BRICKING

As mentioned in section 2, the voxel volume can be divided into chunks, called bricks, in order to cope with voxel data sets sizes exceeding the size of the texture memory of the graphics hardware. Note that our bricks contain the original scalar values of the voxel volume, thus the values *before* applying the transfer function. This enables us to change the transfer function on the fly, since a transfer function change does not require creating new textures.

To obtain a correct interpolation at the bricks' boundaries it is necessary that the data held by adjacent bricks overlap. The overlap depends on the convolution kernel used for interpolation [ML94], and should correspond to (*kernelsize* - 1). For nearest neighbor interpolation that means that no overlap is needed, since the width of the kernel is one. For tri-linear interpolation the overlap should be one voxel in every direction (for other kernels the overlap may even be larger). Pre-integrated rendering [EE02, EKE01, RGW+03] or the on-the-fly calculation of gradients require the overlap to be increased by another voxel in every direction. For bricks of b^3 voxels and an overlap of n voxels, the memory overhead is approximately $(3n/b) \cdot 100\%$.

The bricks are loaded into the video memory as 3D textures. Many graphics cards require 3D texture sizes to be a power of 2 in every direction. If the volume dimensions do not divide evenly into brick dimensions, either an additional layer of partially empty bricks should be added in each direction, or smaller rest-bricks should be used.

When the amount of data in the textures exceeds the available texture memory, textures are swapped between the main memory and the texture memory. If a requested brick is not resident in the texture memory, it is loaded from the main memory, replacing resident textures [SWND03]. In most OpenGL implementations resident textures are swapped out on a Least Recently Used (LRU) base.

Traditionally bricking in texture based rendering is used to be able to render data sets which exceed the size of the texture memory of the graphics hardware. The bricks are then chosen to be as large as possible, and they are sequentially loaded from the main memory into the texture memory. Which implies that for each frame the entire volume data is transferred over the bus.

In our approach, however, we choose brick sizes which are considerably smaller. The smaller the brick size is, the bigger is the chance of bricks being completely void *after* applying the transfer function, and void bricks do not need to be drawn. Therefore, once they are swapped out of the texture memory, they are never reloaded into the texture memory, and thus the bus bottleneck is alleviated.

We even apply bricking to volumes which completely fit into the texture memory to improve data locality, which will result in less cache trashing on the graphics card [HG97, CBS98, IEP98]. On the other hand smaller bricks could introduce a larger overhead due to the overlap needed for interpolation. Thus the optimal brick size needs to be defined depending on the available texture memory, optimal texture size (see section 3), nature of the data set, overhead due to overlap, and the constraints posed by the graphics hardware.

6. EARLY RAY TERMINATION

To be able to perform early ray termination at all, the volume has to be traversed in a front-to-back order. This can be done by evaluating the volume rendering integral in discrete steps, using the under operator:

$$C_{i+1} = (1 - A_i) \cdot \alpha_i \cdot c_i + C_i$$

$$A_{i+1} = (1 - A_i) \cdot \alpha_i \quad + A_i$$

Whereby C, A denote the color, respectively the opacity value of the current ray, c, α the color and opacity value given by applying the transfer function to the current sample in the volume, and i denotes the

ISBN 1213-6972 ISBN 80-86943-09-7

sample index. A ray is then saturated when A_i approximates 1.

Before a brick is rendered, early ray termination is applied to its destination pixels. This is tested by executing a fragment shader program, while drawing a solid bounding box around the brick with back face culling switched on. The fragment shader program writes the maximal value in the depth buffer for saturated rays [KW03, RGW+03]. When slicing the brick texture the early z-test will prevent any fragment operations to be executed for those rays, reducing the load on the rasterization and fragment shader bottlenecks. Early ray termination is only performed once per brick, and not more often (e.g. for every octree node or every sample) because the overhead involved (changing fragment shaders, performing the test) would otherwise annihilate the benefits.

7. OCTREE

By not rendering the void bricks, the load on the rasterization bottleneck is already reduced. We seek to reduce it further by applying octrees. Every brick possesses its own octree. Every octree node corresponds to a cuboid part of the voxel volume, which can be divided into eight parts, corresponding to the child nodes (see figure 4). Our octree is kept in main memory. It only describes the geometry of the visible data. The actual voxel data is to be found in the brick textures.

For tri-linear interpolation, let a cell be defined as a cube, whose eight corners adjacent voxel values are assigned. For every position within the cell an intensity value is defined as the tri-linear interpolation of the corner values. Therefore a cell can only be completely void if its eight corner values are completely transparent ($\alpha = 0$) *after* applying the transfer function. This definition can easily be extended to any given interpolation kernel, by setting the size of a cell to $(kernelsize - 1)^3$.

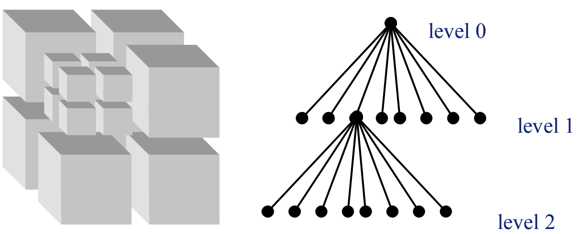


Figure 4: An octree division and its tree.

Every octree node carries a variable describing the ratio r of visible data to total data within its cube. At the final level of the octree, every node represents uniquely one cell, and is considered either completely filled ($r = 1$) or void ($r = 0$). Every higher octree level nodes ratio can be calculated by averaging the ratios of its children. This calculation only needs to be performed when the transfer function has changed.

Rendering an image means that the bricks have to be processed in a front to back order. For each brick the respective octree is traversed, starting with its parent node. Depending on its ratio r there are three ways to process a node:

$r = 0$: The node is completely void. It is not drawn at all, and is not traversed any further.

$0 < r < threshold$: The nodes children will be traversed, and to each child node this strategy will be applied recursively.

$r \geq threshold$: The node is drawn completely. It is not traversed any further.

If the *threshold* is set to 1, exactly all filled cells will be drawn, and no void cells. However, that would lead to a lot of tiny cubes at the boundaries of the visible data structures, and thus the load on the triangle throughput bottleneck becomes too high. Therefore the *threshold* should be chosen in such a way that some degree of void data is allowed to be drawn. A further strategy we use to prevent too much overhead is setting an octree level at which nodes, lower in the hierarchy, are not traversed any further. At this level, any node that is not void, will be drawn completely.

When traversing a node, its children have to be sorted in a front to back order. Since there are eight children, it would seem that there are 8! = 40320 ways to arrange the children. But since the arrangement along the three perpendicular axes is the same for all children, there remain $2^3 = 8$ possible orders. When a node is to be drawn, the cuboid box corresponding to this node is sliced, and the slices are rasterized and blended into the previously drawn slices. The slices can be axis-aligned or viewport-aligned. For the most straight-forward form of volume rendering, the brick texture is interpolated on every slice, taking its position in the brick into account, and after interpolation the transfer function is applied. However, it is also possible to perform more sophisticated forms of volume rendering on the slices, like pre-integrated volume rendering or include specular lighting [MGS02, RGW+03].

The octree is generated and traversed on the CPU. Its purpose is to lower the workload on the graphics pipeline, and thus the GPU. The octree reduces the time that the GPU spends on processing data which never contribute to the final image. The actual volume rendering algorithm, as well as interpolation, the post-interpolative transfer function, and optionally, specular lighting, is being performed by the GPU.

Graphics card	(a) Optimized	(b) Non-optimized	(a) / (b)
nVidia QuadroFX 3400	73.5 fps	9.6 fps	7.66
ATi FireGL X1, xy aligned	83.3 fps	0.23 fps	362
ATi FireGL X1, non xy aligned	27.4 fps	0.23 fps	119
3Dlabs Wildcat 7110	21.3 fps	0.38 fps	56.1

Table 1: Average frame rates reached when using (a) best combination of bricking and octrees, (b) GPU rendering without bricking or octrees.

8. RESULTS

The described approaches have been tested with several different graphics cards: the nVidia QuadroFX 3400 (256MB on board memory), the ATi FireGL X1 (128MB), and the 3DLabsWildcat 7110 (256MB). With each card the volume in figure 3b has been rendered, using the same lookup table settings. The volume data concerned the iliac vein, acquired through 3D rotational angiography. Since contrast media was injected into the vein, the vein could easily be classified using the transfer function. Only 3% of the voxels in this volume contain visible data. All results have been obtained, using a view port of 800^2 pixels and the sample rate for the volume rendering equation was set to 1.5 samples per voxel.

Since the optimal brick size is mainly determined by the properties of the texture memory (see section 5) and the optimal octree limit is primarily used to balance the rasterization load and the triangle throughput (see sections 3 and 7) they can be considered to be fairly orthogonal variables. Therefore their optimum can be found by varying one variable, while keeping the other one constant.

On each graphics card the test volume was rendered with different brick sizes, see figure 5, while the octree limit was set to 8^3 voxels.

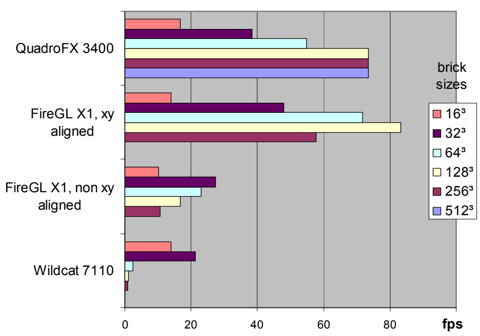


Figure 5: Performance using different brick sizes.

The ATi FireGL X1 and the 3DLabs Wildcat 7110 clearly show that their optimal brick size is considerably smaller than their largest possible brick size. The nVidia QuadroFX 3400 does not benefit from the bricking for the 256MB test volume. However, also this card clearly profits from the bricking for the sparse 1GB volume in figure 3c: the optimal brick size is then 64^3 voxels, with an average frame rate of 37 fps, while for 256^3 bricks only a mere 3.1 fps is reached.

The performance of the ATi FireGL X1 depends heavily on the sampling direction of the bricks, because the ATi card treats the 3D textures as a stack of 2D slices. When the bricks are traversed in the x or y direction, the slices are accessed rather linear, and the performance is much better than when they are traversed in the z direction. It is inevitable to traverse in the z direction, when the viewing direction and the z-axis of the textures differ more than 45°. This effect can be reduced by alternating the orientation of the textures for each consecutive brick [WWE04]. Especially striking is the fact that the optimal brick size and octree limit is different for each sampling direction. When sampled in the xy-plane direction larger bricks benefit from linear traversal, while in other directions smaller bricks benefit from less cache trashing. In figures 5 and 6 this fact is illustrated by the performance measurement when sampling aligned to the xy-plane, and when not.

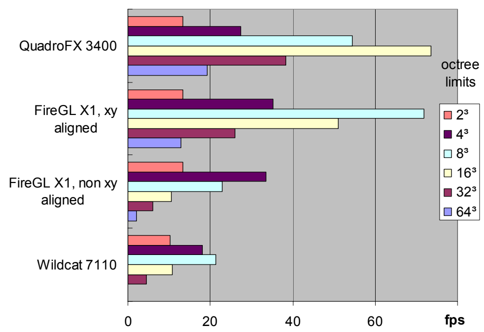


Figure 6: Performance using different octree limits.

Further the volume was rendered with a fixed brick size of 64^3 voxels and variable octree limits (the octree limit is the smallest octree cube allowed). Not every octree branch reaches this limit, see section 7. Figure 6 unsurprisingly shows that there is an optimum octree size for every graphics card. Smaller octree limits lead to too much CPU overhead and triangle count, and larger octrees to too much rasterization overhead. The 64^3 octree level corresponds to not using any octrees at all, only bricking.

Table 1 shows the acceleration achieved, using the volume in figure 3b, with an optimal combination of brick size and octree depth for each particular graphics card versus the same GPU volume rendering routines applied without any bricking or octrees at all. Since early ray termination does not provide any performance gain for sparse data sets, it was not used on this volume.

Early ray termination was tested on the QuadroFX 3400 using the volume in figure 3a. GPU volume rendering without optimizations yielded 2.2 fps, using 64^3 bricks and 8^3 octree limits 5.2 fps were reached, and with additionally early ray termination switched on, the average frame rate was 16.1 fps.

Since the rendering primarily depends on the graphics card, replacing e.g. a Xeon 3.0GHz by a Xeon 1.7GHz delivered approximately the same performance figures. The only part which is bounded by the CPU and main memory performance is building a new octree after the transfer function has been changed. For a volume consisting of 512^3 voxels (16 bit per voxel, 256MB for the entire volume), rendered with a brick size of 64^3 voxels and an octree limit of 8^3 voxels, building all new octrees for the entire 512^3 volume took 6.5 milliseconds on the Xeon 1.7GHz and 3.5 milliseconds on the Xeon 3.0GHz machine.

9. CONCLUSIONS

In this paper, we presented an approach to accelerate GPU-based volume rendering. The approach consisted of a two staged space-skipping and early ray termination, and was tailored to lift the various bottlenecks encountered in the graphics pipeline.

In the first stage, the entire volume is chopped into bricks, and from these bricks 3D textures are created. Empty bricks are never drawn, nor kept in the video memory, and therefore the bus bottleneck is relieved. The optimal brick size depends on the nature of the data (there should be a reasonable chance that there are bricks which are completely void), the available texture memory, the texture cache size and the overhead introduced by brick overlap. Since the brick textures' content does not depend on the transfer function, they need to be created only once for static data.

The octrees, which form the second stage, focus on skipping data that is not visible after applying the transfer function. In this way the rasterization bottleneck is addressed. To prevent too much overhead to be introduced, a certain amount of void data per octree box is allowed, and there is a limit to the granularity of the octree boxes. The optimal octree parameters are determined by the weight of the rasterization phase (i.e. are there complex fragment shader programs involved, etc.) and the trade-off between less rasterization operations and more triangles (triangle throughput bottleneck). Since the octree depends on the transfer function, it has to be recalculated when the transfer function changes.

In this article it has been shown how the individual bottlenecks have been addressed by a two-folded approach. First the bus bottleneck and texture cache size has been addressed by bricking, and consequently the rasterization bottleneck has been addressed by the octrees. The rasterization and fragment shader bottleneck were further lifted by employing early ray termination. The results show that the parameters can be optimized for different graphics cards. Since the transfer function only leads to recalculating the octrees, and not reloading the bricks, it can also be changed quickly and interactively.

The graphics industry are introducing more powerful hardware at an impressive pace. However developments in medical imaging modalities are equally impressive, resulting in larger volume data sets. Which means that in the foreseeable future the techniques that were presented here will preserve their benefits.

10. REFERENCES

[Ake93] K. Akeley. Reality Engine Graphics. In Proc. SIGGRAPH'93, volume 27, pp. 109-116, 1993.

[BNS01] I. Boada, I. Navazo, and R. Scopigno. Multiresolution Volume Visualization with a Texture-based Octree. The Visual Computer, (17), pp. 185-197, 2001.

[CBS98] M. Cox, N. Bhandri, and M. Shantz. Multi-Level Texture Caching for 3D Graphics Hardware. In ISCA '98, pp. 86-97, 1998.

[CCF94] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. Proc. of the 1994 symposium on Volume visualization, pp. 91-98, 1994.

[CN93] T. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, 1993.

[DPH+03] D. E. DeMarle, S. Parker, M. Hartner, C. Gribble, C. Hansen. Distributed Interactive Ray Tracing for Large Volume Visualization. In Proc. 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics, pp. 87-94, 2003.

[Eck98] G. Eckel. OpenGL Volumizer Programmer's Guide. Silicon Graphics, Inc, 1998.

[EE02] K. Engel and T. Ertl. Interactive High-Quality Volume Rendering with Flexible Consumer Graphics Hardware. In Eurographics '02 - State of the Art Report, 2002.

[EKE01] K. Engel, M. Kraus, and T. Ertl. High-quality Pre-integrated Volume Rendering using Hardware-Accelerated Pixel Shading. Proc. of the 2001

Eurographics workshop on Graphics hardware, pp. 9-16, 2001.
[GBKG04] S. Grimm, S. Bruckner, A. Kanitsar and E. Gröller. Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data. IEEE Symposium on Volume Visualization and Graphics, pp. 1-8, 2004.
[GWGS02] S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive Rendering of Large Volume Data Sets. Proc. IEEE Visualization 2002, pp. 53-60, 2002.
[HG97] Z. S. Hakura and A. Gupta. The Design and Analysis of a Cache Architecture for Texture Mapping. In ISCA '97: Proc. of the 24th annual international symposium on Computer architecture, pp. 108-120, 1997.
[IEP98] H. Igehy, M. Eldridge, and K. Proudfoot. Prefetching in a Texture Cache Architecture. In Proc. of the 1998 Eurographics Workshop on Graphics Hardware, pp. 133-142, 1998.
[KKH01] J. Kniss, G. Kindlmann, and C. Hansen. Interactive Volume Rendering using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. Proc. IEEE Visualization 2001, pp. 255-262, 2001.
[KodBA98] R. Kemkers, J. op de Beek, and H. Aerts. 3D-Rotational Angiography: First Clinical Applications. Proc. in Computer Assisted Radiology and Surgery, pp. 182-187, 1998.
[KW03] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In Proc. IEEE Visualization 2003, pp. 287-292, 2003.
[Lev90] M. Levoy. Effcient Ray Tracing of Volume Data. ACM Transactions on Graphics 9(3), pp. 245-261, 1990.
[LHJ99] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In Proc. IEEE Visualization '99, pp. 355-361, 1999.
[MGS02] M. Meissner, S. Guthe, and W. Strasser. Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators. In Graphics Interface 2002, pp. 209-218, 2002.
[ML94] S. R. Marschner and R. J. Lobb. An Evaluation of Reconstruction Filters for Volume Rendering. Proc. IEEE Visualization '94, pp. 100-107, 1994.
[OM01] J. Orchard and T. Möller. Accelerated Splatting using a 3D Adjacency Data Structure. In Graphics interface 2001, pp. 191-200, 2001.
[PSL+99] S. Parker, P. Shirley, Y. Livnat, C. Hansen, P.-P. Sloan, M. Parker. Interacting with Gigabyte Volume Datasets on the Origin 2000. The 41st Annual Cray User's Group Conference , 1999.
[RGW+03] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart Hardware-Accelerated Volume Rendering. In VisSym'03: Proc. of the symposium on Data Visualisation 2003, pp. 231-238, 2003.
[SFH97] R. Srinivasan, S. Fang, and S. Huang. Rendering by Template-based Octree Projection. Proc. of the 8th Eurographics Workshop on Visualization in Scientific Computing, pp. 155-163. Eurographics, 1997.
[SWND03] D. Shreiner, M. Woo, J. Neider, and T. Davis. OpenGL Programming Guide: The Offcial Guide to Learning OpenGL (red book). Addison-Wesley Pub Co, 4 edition, 2003.
[TWTT99] X. Tong, W. Wang, W. Tsang, and Z. Tang. Efficiently Rendering Large Volume Data Using Texture Mapping Hardware. In Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym), pp. 121-132, 1999.
[vdB03] J. C. van den Berg. Three-Dimensional Rotational Angiography. Endovascular Today, (March 2003), 2003.
[WS01] R. Westermann and B. Sevenich. Accelerated Volume Ray-Casting using Texture Mapping. Proc. IEEE Visualization 2001, pp. 271-278, 2001.
[WWH+00] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-Of-Detail Volume Rendering via 3D Textures. In Proc. Volume Visualization and Graphics Symposium 2000, pp. 7-13, 2000.
[WWE04] D. Weiskopf, M. Weiler, T. Ertl. Maintaining Constant Frame Rates in 3D Texture-Based Volume Rendering. Computer Graphics International 2004 (CGI'04), pp. 604-607, 2004.
[YS93] R. Yagel and Z. Shi. Accelerating Volume Animation by Space-Leaping. Proc. IEEE Visualization '93, pp. 62-69, 1993.
[Zel02] C. Zeller. Balancing the Graphics Pipeline for Optimal Performance, Graphics Developer Conference 2002, http://developer.nvidia.com/, 2002.
[ZKV92] K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Accelaration of Ray-Casting Using 3D Distance Transform. Proc. of Visualization in Biomedical Computing II, pp. 324-335, 1992.

Hardware-Accelerated Collision Detection using Bounded-Error Fixed-Point Arithmetic

Andreas Raabe,[1] Stefan Hochgürtel,[1] Joachim K. Anlauf,[1] Gabriel Zachmann [2]

[1]Technical Computer Science
Bonn University, Germany
{raabe, hochguer, anlauf}@cs.uni-bonn.de

[2]Computer Graphics
Clausthal University, Germany
zach@in.tu-clausthal.de

ABSTRACT

A novel approach for highly space-efficient hardware-accelerated collision detection is presented. This paper focuses on the architecture to traverse bounding volume hierarchies in hardware. It is based on a novel algorithm for testing discretely oriented polytopes (DOPs) for overlap, utilizing only fixed-point (i.e., integer) arithmetic. We derive a bound on the deviation from the mathematically correct result and give formal proof that no false negatives are produced.

Simulation results show that real-time collision detection of complex objects at rates required by force-feedback and physically-based simulations can be obtained. In addition, synthesis results prove the architecture to be highly space efficient. We compare our FPGA-optimized design with a fully parallelized ASIC-targeted architecture and a software implementation.

1 INTRODUCTION

Detecting collisions between a pair of graphical objects is a fundamental task in many areas such as physically-based simulation, automatic path finding, or tolerance checking. Applications are in games, animation systems, and virtual reality, e.g., virtual assembly simulation, or medical training and planning systems.

In most of the applications in these areas, the goal is to avoid collisions, or to enable real-time physically-based simulation. Most approaches today are reactive, i.e., they first place objects at a new trial position, check for collisions, and then compute new forces or positions, based on physical laws, so as to remove any collisions.

This approach demands very efficient collision detection, because it must perform many collision checks per simulation cycle. An emerging application area is the mobile devices market (smart phones, portable games devices). Here, the challenges, besides speed, are size and energy consumption. Another particularly demanding application is force-feedback, where updates of about 1000Hz must be done in order to achieve stable force computations.

Since collision detection is such a fundamental yet challenging task, it is highly desirable to have hardware acceleration available just like 3D graphics accelerators. The benefit is two-fold: a) the system can process objects with higher polygon counts, and b) the system's CPU can be freed from computing collisions.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

In this paper, we present a novel, efficient architecture for hierarchical collision detection of two rigid objects. It is based on a novel algorithm for testing a pair of bounding volumes for overlap, which can even be implemented on fixed-point arithmetic. We derive a tight bound on the deviation between overlap testing on fixed-point vs. floating-point arithmetic; this ensures that no false negatives are obtained while producing very few false positives.

We also present an implementation on FPGA hardware along with simulation results concerning its speed and synthesis results concerning its size. Finally, we compare these results with an earlier, parallelized ASIC-targeted architecture, and with a software implementation.

2 RELATED WORK

Considerable work has been done on hierarchical collision detection in software [3, 4, 8, 14, 15]. Some of the bounding volumes (BVs) utilized are spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), and discretely oriented polytopes (DOP).

The first work on dedicated hardware for collision detection was presented in [16, 17]. However, they presented only a functional simulation, while we present a RT level implementation along with synthesis results. [12] presented a design that was targeted on ASICs, and was optimized for speed only, and, thus, utilize a total of over 4 million gates and a 756 bits wide bus to a DDR2-RAM. Recently, a commerical hardware was announced that supposedly can do collision detection, among other things [1]. However, no details have been published, in particular, no performance results.

Most other hardware-related research has tried to utilize existing graphics accelerator boards (GPU) [2, 5, 6, 10, 11]. While earlier approaches, such as [11], can basically handle only convex objects, later algorithms, such as [2, 10], have extended these to more general cases such as unions of convex objects or closed objects. The general class of "polygon soups" can be handled by [5], but they use a hybrid approach where the graphics hardware only identifies potentially colliding sets.

All of the approaches using graphics hardware have the disadvantage that they either compete with the rendering process for the same hardware resource, or an additional graphics board must be spent for collision detection. The former slows down the overall frame rate considerably, while the latter would be a tremendous waste, since most of the resources of the hardware would not be utilized at all. Furthermore, most of these approaches work in image space, which reduces precision significantly.

3 BASICS

3.1 Hierarchical Collision Detection with *k*-DOPs

In this section, we give a short outline of the algorithm of hierarchical collision detection in conjunction with a special kind of bounding volumes, the *k*-DOP. Additionally, we will quickly recap the separating axis theorem and one of its application, the Separating Axis Test (SAT).

In this paper hierarchical collision detection is utilized to avoid checking every triangle of an object O for intersection with all triangles of object Q. The acceleration data structure is a so-called *bounding volume hierarchy* (BVH), where each leave corresponds to one triangle and inner nodes correspond to groups of triangles. Each node has a bounding volume (BV) attached that bounds all triangles associated with it. In order to achieve a feasible hardware design, we use a binary tree here.

If two objects are checked for overlap, both hierarchies are traversed simultaneously. If their BVs intersect, the next level of BVs is checked. Since two objects will usually intersect only in a very small number of primitives, this yields a significant speed-up in the average case. In practical cases, the complexity is in $O(\log n)$ (n = number of primitives) because only a small diagonal "slice" of constant width down the BVH needs to be visited [9].

In this work, we use *k*-DOPs as BVs because they were proven to yield very fast collision queries by extensive benchmarking in software [15], and performed very well in our hardware studies [12], too.

k-DOPs are defined over a fixed orientation matrix $\mathbf{D} = \left(\mathbf{D}_1, \ldots, \mathbf{D}_{k/2}, \mathbf{D}_{k/2+1}, \ldots, \mathbf{D}_k\right)$ of vectors in $\mathbb{R}^3$. Each vector $\mathbf{D}_i$ is antiparallel to $\mathbf{D}_{i+k/2}$.

An individual *k*-DOP is defined by k distances d_i, one along each vector $\mathbf{D}_i$, thus defining a half-space. These DOP coefficients $(d_1, \ldots, d_k)$ are the distances of the associated halfspaces to the origin.[1] The $k/2$ pairs of DOP coefficients $(d_i, d_{i+k/2})$ form a so-called *slab* [15].

The intersection of these slabs forms the BV:

$$\text{DOP} = \bigcap_{i=1,\ldots,k} H_i, \quad H_i : \mathbf{D}_i\mathbf{x} - d_i \leq 0 \qquad (1)$$

The orientation matrix D, consisting of all the vectors $\mathbf{D}_i$, is fixed and equal for all objects. This yields very memory-efficient description for every *k*-DOP: only the k coefficients d_i need to be stored.

3.2 Separating Axis Test (SAT)

In this paper we use the so called Separating Axis Test (SAT) introduced by [4, 14].

[4] have shown that two convex polytopes are disjoint if and only if there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope (Separating Axis Theorem).

If only a subset of these axes are tested, false positives might occur, i.e., the polytopes are disjoint while the (incomplete) test yields an intersection. The complete SAT is always correct.

To perform the test, both polytopes must be projected onto each of the candidate separating axes. For each axis, a pair of intervals on that axis results. If one of these pairs is disjoint, then the polytopes must be disjoint (see Fig. 1).

4 EFFICIENT SAT FOR *K*-DOPS

In this section, we will derive an efficient Separating Axis Test for *k*-DOPs. Additionally, we will show how the resulting overlap test can be done in fixed-point arithmetic such that no false negatives occur. Finally, we will derive a bound on the deviation of the projection of the fixed-point DOP with respect to the mathematically correct image.

4.1 Precomputation

Since with DOPs the set of vectors $\{D_1, \ldots, D_k\}$ is fixed, we can exploit that all possible face orientations of the DOPs within a DOP-tree are the same.

Assume object O is placed relatively to object Q by rotation $\mathbf{M}$ and translation $\mathbf{T}$. Let $\text{DT}(O)$ and $\text{DT}(Q)$ denote the DOP-trees of these objects. As described in Section 3.1, let $(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ be the orientations of the DOPs' faces shared by all DOPs in $\text{DT}(O)$ after applying rotation $\mathbf{M}$. Analogously, let $(a_1, \ldots, a_k)$ denote the DOP coefficients for DOPs in $\text{DT}(O)$, let $(\mathbf{B}_1, \ldots, \mathbf{B}_k)$ denote the vectors shared by all DOPs in $\text{DT}(Q)$, and let $(b_1, \ldots, b_k)$ denote the corresponding DOP coefficients.

[1] Note that the origin is not necessarily the center of the DOP nor even contained in it.

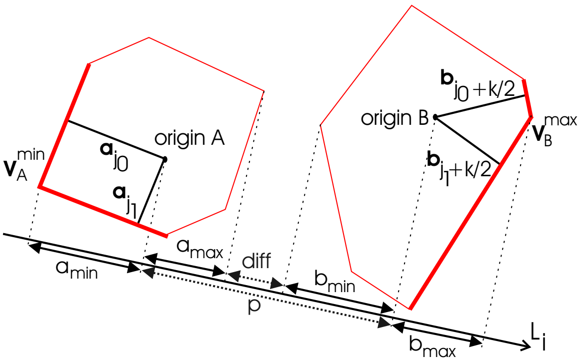


Figure 1: Two DOPs are projected onto test axis L_i. Since their images do not intersect L_i is a separating axis.

Note that everything independent of $(a_1,\ldots,a_k)$ and $(b_1,\ldots,b_k)$ is constant throughout the whole DOP-trees. Hence it can be precalculated at startup to initialize the algorithm (and, later-on, the hardware). Precomputing as much as possible significantly reduces the resulting hardware costs. Since this is done only once per pair of DOP-trees, it is not time-critical.

First, we can precompute the n test axes $\mathbf{L}_i$. All of the following is done for each $\mathbf{L}_i$, so for the sake of simplicity we omit the index i from now on.

Second, the projection $p = \mathbf{L} \cdot \mathbf{T}$ is precomputed.

Third, for each $\mathbf{L}$ a DOP has two vertices $\mathbf{v}_A^{\min}$ and $\mathbf{v}_A^{\max}$ whose projections onto $\mathbf{L}$ have maximum distance. Each of those vertices is formed by the intersection of three faces of the DOP. The correspondences $(j_{A,0}, j_{A,1}, j_{A,2})$ of the orientations whose faces meet in $\mathbf{v}_A^{\min}$ are calculated.

Fourth, and most importantly, in the actual projection

$$a_{\min} = \mathbf{v}_A^{\min} \cdot \mathbf{L}$$
$$= \left(a_{j_{A,0}}\ a_{j_{A,1}}\ a_{j_{A,2}}\right) \cdot \left(\mathbf{A}_{j_{A,0}} \quad \mathbf{A}_{j_{A,1}} \quad \mathbf{A}_{j_{A,2}}\right)^{-1} \cdot \mathbf{L}$$

we can precompute the last dot product

$$\mathbf{P}_A := \left(\mathbf{A}_{j_{A,0}} \quad \mathbf{A}_{j_{A,1}} \quad \mathbf{A}_{j_{A,2}}\right)^{-1} \cdot \mathbf{L} \tag{2}$$

$\mathbf{P}_B$ can be precomputed analogously. The mapping vectors for $\mathbf{v}_A^{\max}$ and $\mathbf{v}_B^{\max}$ are $-\mathbf{P}_A$ and $-\mathbf{P}_B$ respectively. This exploits that $k/2$ pairs of DOP orientations are anti-parallel. Note that this is an estimate to the correct solution, since not all possible combinations of DOP-coefficients share all maximum vertices. But it is impossible for any vertex made-up of the intersection of three faces to be inside the DOP, hence only false positives can result.

4.2 Intersection Testing

Using these precomputations, we can project onto the test axes very efficiently:

$$\begin{aligned} a_{\min} &= \left(\mathbf{a}_{j_{A,0}} \quad \mathbf{a}_{j_{A,1}} \quad \mathbf{a}_{j_{A,2}}\right) \cdot \mathbf{P}_A \\ a_{\max} &= \left(\mathbf{a}_{j_{A,0}+k/2} \quad \mathbf{a}_{j_{A,1}+k/2} \quad \mathbf{a}_{j_{A,2}+k/2}\right) \cdot (-\mathbf{P}_A) \end{aligned} \tag{3}$$

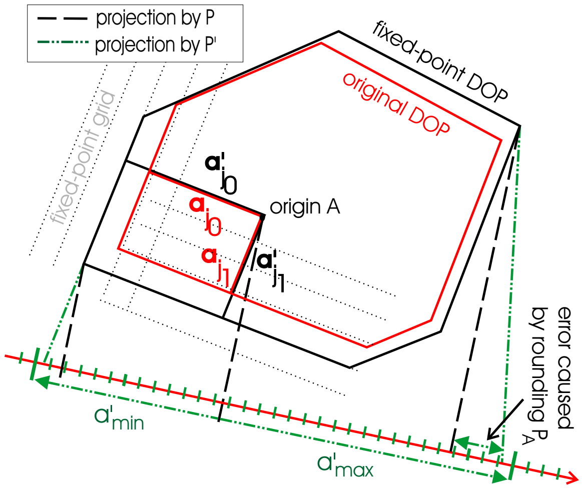


Figure 2: A DOP and its enclosing fixed-point equivalent. Both rounding the DOP to fixed-point numbers and projecting it with $\mathbf{P}'$ instead of $\mathbf{P}$ increases the DOP's image. When checked for intersection false positives can occur.

This is done for $b_{\min}$ and $b_{\max}$ analogously.

The condition for separation is straight-forward now. Let

$$\begin{aligned} \text{diff}_1 &:= (a_{\min} + p) - b_{\max} \\ \text{diff}_2 &:= b_{\min} - (a_{\max} + p) \end{aligned} \tag{4}$$

$$\text{diff} := \max(\text{diff}_1, \text{diff}_2) \tag{5}$$

then the intervals $[a_{\min}, a_{\max}]$ and $[b_{\min}, b_{\max}]$ are disjoint if and only if $\text{diff} > 0$. And from the Separating Axis Theorem we know that

$$(\text{diff} > 0) \Rightarrow \text{separation}. \tag{6}$$

Eqs. (3)–(6) show the computations that need to be done for each DOP test (and hence cannot be precomputed).

4.3 Fixed-Point Arithmetic

In hardware floating-point arithmetic is very expensive with respect to circuit size and depth. Unfortunately, simply rounding DOP coefficients to fixed-point numbers would result in false negatives, because the intervals on the test axes could become smaller than the projection of the enclosed object. These false negatives are inacceptable, because we might miss collisions. Naïve rounding of the mapping vectors $\mathbf{P}_A$ and $\mathbf{P}_B$ would lead to even more false negatives since distance of the images could be overestimated. Hence we need to round in a manner such that each fixed-point DOP image contains the according floating-point image (see Fig. 2).

First, we need to handle the smaller scale of fixed-point numbers by dividing all DOP coefficients of all DOPs by the largest absolute value of the DOP coefficients in the scenario. This way, 16 bit accuracy still allows for having DOPs the size of a skyscraper and of

 ISBN 1213-6972 ISBN 80-86943-09-7

a 6mm screw. 36 bit even allow for DOPs the size of the sun and of a screw.

Let rounding of the DOP coefficients to b bits after the point towards $+\infty$ be denoted by $a'_i = \lceil a_i \rceil$. Clearly, the rounded (i.e., fixed-point) DOP contains the original one. Then, $\varepsilon_i = a'_i - a_i$ is the resulting rounding error, with $0 \le \varepsilon_i < 2^{-b}$.

By ensuring that the dihedral angle between all pairs of neighboring faces of a DOP is larger than $\pi/2$, all $P_{A,i}$ are in the interval $[-1,0]$ [7].[2]

Rounding $\mathbf{P}_{A,i}$ towards $-\infty$ to c bit accuracy results in a rounding error $0 \le \eta_i = \mathbf{P}_{A,i} - \mathbf{P}'_{A,i} < 2^{-c}$.

By simply truncating $\mathbf{P}_{A,i}$, the resulting image would become too small in case of negative DOP coefficients, whereas always rounding up would create the same problem with positive coefficients. Fortunately, we can solve this during calculation simply by adding 2^{-c} to $\lfloor \mathbf{P}_{A,i} \rfloor$ before multiplication with negative DOP coefficients.

Let $\mathbf{a}' := (a'_{j_{A,0}}, a'_{j_{A,1}}, a'_{j_{A,2}})$ and $\mathbf{a}'_k := (a'_{j_{A,0}+k/2}, a'_{j_{A,1}+k/2}, a'_{j_{A,2}+k/2})$. Let $\mathrm{sn}(\mathbf{x})$ be the sum of all $\mathbf{x}_i < 0$.

Then, correct rounding of the images amounts to:

$$\begin{aligned} a'_{\min} &= \mathbf{P}'_A \cdot \mathbf{a}' + 2^{-c}\,\mathrm{sn}(\mathbf{a}') \\ a'_{\max} &= -(\mathbf{P}'_A \cdot \mathbf{a}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k)) \end{aligned} \tag{7}$$

Finally, when computing diff_1, we can simply truncate p to z bits ($p' = \lfloor p \rfloor$). This can create only false positives, because a smaller p' only decreases the apparent distance between the two DOP images. For diff_2 we need to round p up to $\lceil p \rceil$, which, again, can be done efficiently by adding 2^{-z} to $\lfloor p \rfloor$.

Overall, calculating the distances of the fixed-point DOP images amounts to

$$\begin{aligned} \mathrm{diff}'_1 &= (\mathbf{a}'_{\min} + p') - \mathbf{b}'_{\max} \\ \mathrm{diff}'_2 &= \mathbf{b}'_{\min} - (\mathbf{a}'_{\max} + (p' + 2^{-z})) \end{aligned} \tag{8}$$

$$\mathrm{diff}' = \max(\mathrm{diff}'_1, \mathrm{diff}'_2) \tag{9}$$

Now the condition for separation can be given analogously to Eq. 6:

$$((\mathrm{diff}'_1 > 0) \text{ or } (\mathrm{diff}'_2 > 0)) \Rightarrow \text{separation.} \tag{10}$$

Simulations done early in the design process showed that fixed-point accuracy influences calculation time (Fig. 3). Below 18 bits accuracy, an increasing number of false positives occurs compared to the floating-point implementation and decreases calculation speed. Above 18 bits, a second memory burst is needed to fetch DOP coefficients from DDR-RAM.

[2] This is no hard restriction since every well-constructed DOP should not have acute angles to improve tightness of fit (even for oblong objects in random orientation).

4.4 Bound on Fixed-Point Deviation

In this section we will derive a bound on the deviation of the fixed-point image from the mathematically correct image. Let err denote this deviation (called *fixed-point error* in the following)

$$\mathrm{err} := \mathrm{diff} - \mathrm{diff}' \tag{11}$$

Since diff is defined as $\max(\mathrm{diff}_1, \mathrm{diff}_2)$ (and diff' analogously) we know that

$$\begin{aligned} \mathrm{err}_1 &:= \mathrm{diff}_1 - \mathrm{diff}'_1 \\ \mathrm{err}_2 &:= \mathrm{diff}_2 - \mathrm{diff}'_2 \end{aligned} \tag{12}$$

$$\min(\mathrm{err}_1, \mathrm{err}_2) \le \mathrm{err} \le \max(\mathrm{err}_1, \mathrm{err}_2) \tag{13}$$

Inserting Eqs. (3)–(5) and (7)–(9) into Eq. (12) yields

$$\begin{aligned} \mathrm{err}_1 = &(\mathbf{P}_A \cdot \mathbf{a} - \mathbf{P}'_A \cdot \mathbf{a}') - 2^{-c} \cdot \mathrm{sn}(\mathbf{a}') \\ &+ (\mathbf{P}_B \cdot \mathbf{b}_k - \mathbf{P}'_B \cdot \mathbf{b}'_k) - 2^{-c} \cdot \mathrm{sn}(\mathbf{b}'_k) \\ &+ (p - p') \end{aligned} \tag{14}$$

and err_2 can be calculated analogously.

To calculate bounds on err_1 and err_2 we need to bound the errors caused by products of mapping vectors and DOP coefficients. Since this is all very similar, we show how it is done, for example, for $\mathbf{P}_A \cdot \mathbf{a} - \mathbf{P}'_A \cdot \mathbf{a}'$.

$$\begin{aligned} &\mathbf{P}_A \cdot \mathbf{a} - \mathbf{P}'_A \cdot \mathbf{a}' \\ =&(\mathbf{P}_A - \mathbf{P}'_A) \cdot \mathbf{a}' + \mathbf{P}_A \cdot (\mathbf{a} - \mathbf{a}') \\ =&\sum_{i=0}^{2} (\mathbf{P}_{A,i} - \mathbf{P}'_{A,i}) \cdot \mathbf{a}'_{j_{A,i}} + \sum_{i=0}^{2} \mathbf{P}_{A,i} \cdot (\mathbf{a}_{j_{A,i}} - \mathbf{a}'_{j_{A,i}}) \\ =&\sum_{\substack{i=0 \\ a'_{j_{A,i}} \ge 0}}^{2} (\mathbf{P}_{A,i} - \mathbf{P}'_{A,i}) \cdot \mathbf{a}'_{j_{A,i}} + \sum_{\substack{i=0 \\ a'_{j_{A,i}} < 0}}^{2} (\mathbf{P}_{A,i} - \mathbf{P}'_{A,i}) \cdot \mathbf{a}'_{j_{A,i}} \\ &+ \sum_{i=0}^{2} \mathbf{P}_{A,i} \cdot (\mathbf{a}_{j_{A,i}} - \mathbf{a}'_{j_{A,i}}) \end{aligned} \tag{15}$$

The cross sum of any mapping vector can be interpreted as the image of a vertex of the maximum DOP (all $d_i = 1$). Assume r_{max} to be the greatest distance and $r_{min} = 1$ to be the smallest distance of a vertex of the maximum DOP to the origin. Then $-r_{max}$ is a lower bound and $-r_{min}$ is an upper bound for the cross sum of any mapping vector $\mathbf{P}$.

Let $\mathrm{sp}(\mathbf{x})$ be the sum of all $\mathbf{x}_i \ge 0$ analogously to $\mathrm{sn}(\mathbf{x})$, the sum of all $\mathbf{x}_i < 0$.

With the known boundaries

$$\begin{aligned} &-1 \le \mathbf{a}'_i \le 1 && -2^{-b} \le \mathbf{a}_i - \mathbf{a}'_i \le 0 \\ &-1 \le \mathbf{P}_{A,i} \le 0 && 0 \le \mathbf{P}_{A,i} - \mathbf{P}'_{A,i} \le 2^{-c} \\ & && 0 \le p - p' \le 2^{-z} \end{aligned}$$

 ISBN 1213-6972 ISBN 80-86943-09-7

we can bound all summands of Eq. (15):

$$0\cdot \mathrm{sp}(\mathbf{a}') \le \sum_{\substack{i=0\\ a'_{j_{A,i}}\ge 0}}^{2} (\mathbf{P}_{A,i}-\mathbf{P}'_{A,i})\cdot \mathbf{a}'_{j_{A,i}} \le 2^{-c}\cdot \mathrm{sp}(\mathbf{a}')$$

$$2^{-c}\cdot \mathrm{sn}(\mathbf{a}') \le \sum_{\substack{i=0\\ a'_{j_{A,i}}< 0}}^{2} (\mathbf{P}_{A,i}-\mathbf{P}'_{A,i})\cdot \mathbf{a}'_{j_{A,i}} \le 0\cdot \mathrm{sn}(\mathbf{a}')$$

$$-r_{min}\cdot 0 \le \sum_{i=0}^{2} \mathbf{P}_{A,i}\cdot(\mathbf{a}_{j_{A,i}}-\mathbf{a}'_{j_{A,i}}) \quad \le -r_{max}\cdot(-2^{-b})$$

Along with Eq. (15) this amounts to

$$2^{-c}\cdot \mathrm{sn}(\mathbf{a}') \le \mathbf{P}_A\cdot\mathbf{a} - \mathbf{P}'_A\cdot\mathbf{a}' \le 2^{-c}\cdot\mathrm{sp}(\mathbf{a}') + r_{max}\cdot 2^{-b} \tag{16}$$

Inserting Eq. (16) into Eq. (14) yields

$$\begin{aligned}\mathrm{err}_1 \le\; & 2^{-c}\cdot\mathrm{sp}(\mathbf{a}') + r_{max}\cdot 2^{-b} - 2^{-c}\,\mathrm{sn}(\mathbf{a}') \\ & +2^{-c}\cdot\mathrm{sp}(\mathbf{b}'_k) + r_{max}\cdot 2^{-b} - 2^{-c}\,\mathrm{sn}(\mathbf{b}'_k) + 2^{-z}\end{aligned} \tag{17}$$

and

$$\begin{aligned}\mathrm{err}_1 \ge\; & 2^{-c}\cdot\mathrm{sn}(\mathbf{a}') - 2^{-c}\,\mathrm{sn}(\mathbf{a}') \\ & +2^{-c}\cdot\mathrm{sn}(\mathbf{b}'_k) - 2^{-c}\,\mathrm{sn}(\mathbf{b}'_k) + 0 = 0\end{aligned} \tag{18}$$

Calculating bounds on err_2 can be done analogously and results in

$$\begin{aligned}\mathrm{err}_2 \le\; & 2^{-c}\cdot\mathrm{sp}(\mathbf{b}') + r_{max}\cdot 2^{-b} - 2^{-c}\,\mathrm{sn}(\mathbf{b}') \\ & +2^{-c}\cdot\mathrm{sp}(\mathbf{a}'_k) + r_{max}\cdot 2^{-b} - 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k) - 0 + 2^{-z}\end{aligned} \tag{19}$$

and

$$\begin{aligned}\mathrm{err}_2 \ge\; & 2^{-c}\cdot\mathrm{sn}(\mathbf{b}') - 2^{-c}\,\mathrm{sn}(\mathbf{b}') \\ & +2^{-c}\cdot\mathrm{sn}(\mathbf{a}'_k) - 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k) - 2^{-z} + 2^{-z} = 0\end{aligned} \tag{20}$$

Since we ensured that the dihedral angles between all pairs of neighboring faces exceed $\pi/2$, r_{max} is bounded by $\sqrt{3}$ [7]. Combining this with Eqs. (17)–(20) and inserting the result in Eq. (14) yields the overall result

$$0 \le \mathrm{err} \le \sqrt{3}\cdot 2^{-b+1} + 6\cdot 2^{-c} + 2^{-z} \tag{21}$$

This gives a bound on the deviation of the image size of a fixed-point DOP with respect to the exact image. Additionally, it formally proves that no false negatives can occur.

5 THE ARCHITECTURE

5.1 The Pipeline

Combining Eqs. (7)–(10) results in the overlap condition

$$\begin{gathered}\mathbf{P}'_A\cdot\mathbf{a}' + 2^{-c}\,\mathrm{sn}(\mathbf{a}') + \mathbf{P}'_B\cdot\mathbf{b}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{b}'_k) + p' > 0 \\ \text{or} \\ \mathbf{P}'_B\cdot\mathbf{b}' + 2^{-c}\,\mathrm{sn}(\mathbf{b}') + \mathbf{P}'_A\cdot\mathbf{a}'_k + 2^{-c}\,\mathrm{sn}(\mathbf{a}'_k) \text{-} (p' + 2^{-z}) > 0 \\ \Rightarrow \text{separation}\end{gathered} \tag{22}$$

Eq. (22) is divided into seven stages to enable pipelining.

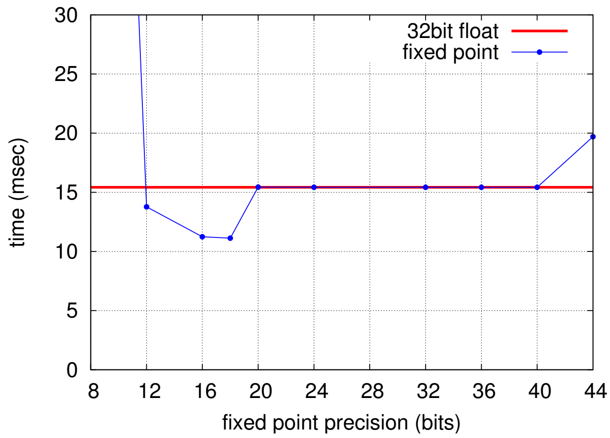


Figure 3: Speed of fixed-point arithmetic for different bit widths. Beyond 18 bits a second, and beyond 40 bits a third memory burst is needed.

Selection. Stage one selects the 12 out of k DOP coefficients defining the outer (maximal) vertices for a given candidate separating axis based on the correspondences (j_A, j_B).[3] Correspondences j_A and j_B contain the indices of 6 of them. The indices of the 6 remaining ones can be derived by simply increasing these indices each by $k/2$. Since we assume wrap around indexing here, this does not need any combinational hardware, but can be done by simply feeding the coefficients into the multiplexers in modified order.

Scalar Products and Fixed-Point Correction. Stages two to five implement the calculation of the scalar products and the fixed-point correction term. So, DOP coefficients have to be multiplied by $\mathbf{P}'$-vector entries and summed up by an adder tree. Additionally, p' ($-(p'+2^{-z})$ in case of diff'_2) is added. Concurrently, negative DOP coefficients are selected and accumulated. Stage six adds the results of both summations. Multiplying by 2^{-c} is done implicitly by shifting.

Result. Testing $\max(\mathrm{diff}'_1, \mathrm{diff}'_2) > 0$ is done by negating the conjunction of the sign bits.

5.2 Overall Design

The overall architecture is shown in Fig. 4. The calculation is initialized by the host system by sending $(\mathbf{P}'_A, \mathbf{P}'_B, p', j_A, j_B)$ and the addresses of the DOP-trees to the hardware. A controller keeps track of DOP overlap tests that must still be executed and requests the needed DOP coefficients and triangle data. The module "GetData" reads them from memory concurrently to the current calculation. As soon as the parameters are loaded and the last calculation is finished, it feeds them into the pipeline (or the triangle-unit respectively). The pipeline receives not only the DOP coefficients but (from the controller) the data for the next axis test.

[3] There are 2 k-DOPs, 2 maximal vertices per DOP, and 3 coefficients defining each vertex (see Section 4.1).

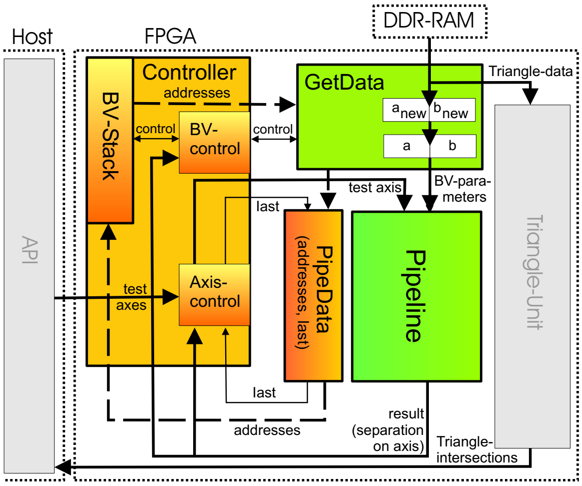


Figure 4: The complete intersection test hardware.

For each DOP pair, n axes are tested. A shift register ("PipeData") holds additional bookkeeping information. For every pipeline stage it contains the indices of the processed DOPs and whether the contained calculation is the last axis test to be executed for the current DOP pair. If this last axis test leaves the pipeline and none of the test axes is a separating axis the controller schedules the child DOPs to be tested. If a separating axis is found, the remaining calculations belonging to the same DOP-pair are obsolete. No new axis tests are initiated and the results of the calculations that are still in the pipeline will be ignored; no new DOP tests are scheduled.

[13] showed that scheduling DOP tests in a stack is far superior to queue control with regards to memory usage. So, as soon as the stack, pipeline, and the GetData module are empty, and no intersecting triangles were found, the objects do not intersect and this is reported to the host application. On the other hand, every intersecting pair of triangles is reported to the host immediately.

To check triangles for intersection we utilize the same algorithm that was already proposed in [16] and implemented in VHDL in [13]. It transforms both triangles so that one of them becomes the “unit” triangle. That way, the checks to be performed on the other triangle become very simple and standardized.

5.3 Control

As mentioned in Section 3.2, it is not necessary to test all axes L_i whether they are separating axes. Even more, [14] has shown that it is not efficient to test all axes for OBBs since the probability of the BV to be disjoint decreases rapidly with every non-separating test axis found so far. Fig. 5 shows that this applies for DOPs, too.

On the other hand, we want to eliminate disjoint branches of the DOP trees as early as possible to reduce expensive loading of DOP coefficients. Therefore,

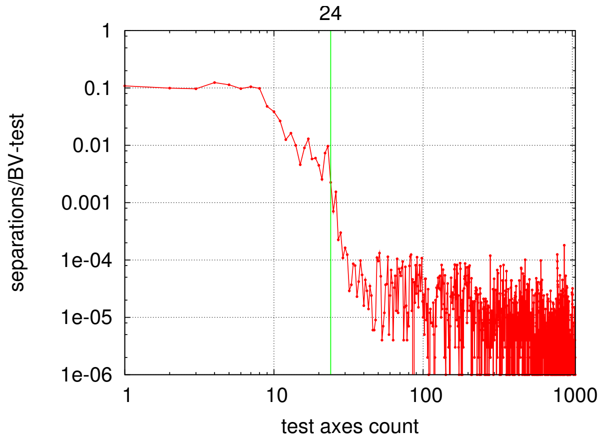


Figure 5: The more axes are tested for intersection the less probable it is for other axes to be separating.

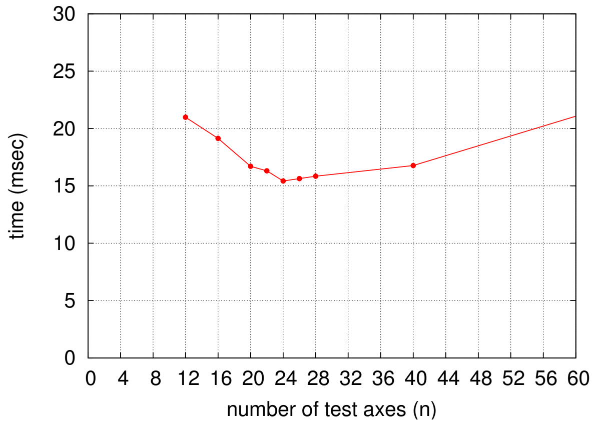


Figure 6: For fixed $k = 24$, our design performs best using $n = 24$ on the target architecture.

we determined which $n \leq N$ gives the best trade-off between axis-testing and parameter-loading. As Fig. 6 indicates, $n = 24$ yields the optimum performance for 24-DOPs and the given memory architecture. 24 axis-tests suffice to test all candidate separating axes generated from the 12 face-orientations of each DOP. Although this exceeds the time to load a complete set of DOP-coefficients (only 20 clock-cycles) by 4 cycles, testing 24 axes seems to reduce the number of false positives enough to yield a performance gain.

Still, there is no reason to stop testing axes if the next DOP-pair is not completely loaded yet. This can happen, for instance, if the memory subsystem is occupied by triangle data. As shown in Fig. 7 continued axes testing until the next set of DOP-coefficients is fetched from memory speeds-up calculation.

6 RESULTS

The target architecture is a Xilinx Virtex II (XC 2V6000, speed grade -4) on an Alpha Data ADM-XRC-II board with 256 MB DDR-RAM at 100MHz connected via a 64 bit wide bus. The FPGA features 144 18-bit multipliers and 6 million gate equivalents. CoCentric from

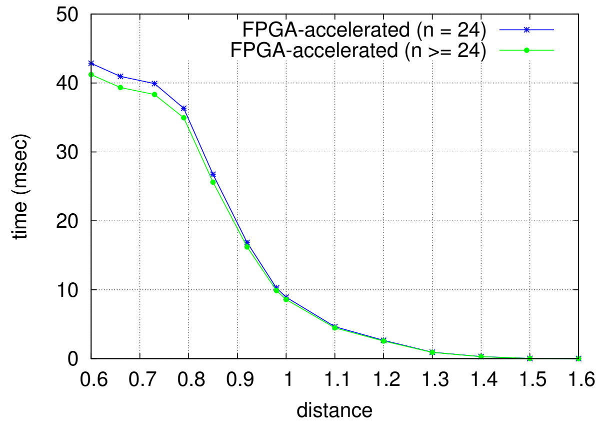


Figure 7: Testing further axes until next DOP-pair is loaded yields a speed-up.

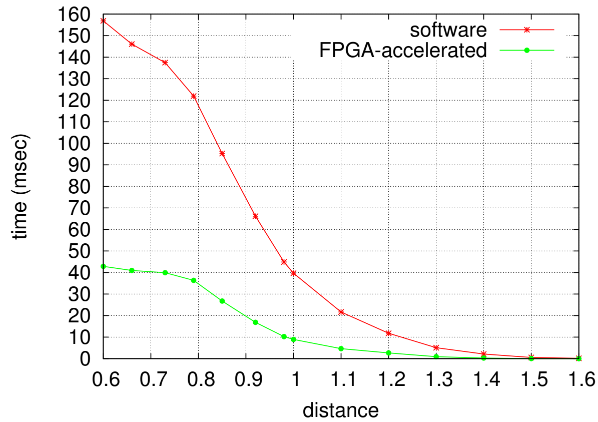


Figure 8: The presented architecture is approximately 4 times faster than a state-of-the-art software intersection test.

Synopsys was used to compile SystemC RTL to VHDL code. Synthesis, Place, Route and Mapping were done with Xilinx ISE 6.3.

6.1 Synthesis Results

Although 19-bit accuracy performs best on our test data with respect to calculation time (Fig. 3), we decided to implement the pipeline for 35 bits fixed-point 24-DOPs to tolerate bigger differences in DOP size (see Section 4.3). Since the target architecture features 18-bit multipliers only, this results in two extra pipeline stages to implement 35-bit pipelined multipliers.

Overall, the pipeline utilizes a total of 7278 out of 33792 slices ($21\% = 1,260,000$ million gate equivalents). Maximum clock frequency is 111.117MHz.

6.2 Benchmarking

All results presented here were obtained with two identical objects (a car headlight) with 5947 triangles [13]. They are placed at different distances from each other and with different rotations. For each constellation, the time to detect all intersecting triangles is determined. Fig. 8 shows the comparison of our new architecture with a state-of-the-art software intersection test running on a 1 GHz Pentium III with 512 MByte main memory. Memory bandwidth and speed are identical on both systems and hence allow for a direct performance comparison. The presented acceleration hardware yields a speed up of about factor 4.

7 CONCLUSION AND FUTURE WORK

We have presented a novel algorithm for hierarchical collision detection of pairs of virtual objects. We have also presented a highly space-efficient, FPGA-optimized architecture implementing this algorithm on an FPGA using fixed-point arithmetic. The fixed-point calculations do not produce any false negatives, and we have given bounds on the deviations from floating-point arithmetic.

Simulation results for collision queries using this architecture proved that a speed-up of 4 compared to state-of-the-art software intersection tests on a standard CPU can be obtained. Taking earlier ASIC-targeted results into account [13], we conclude that an ASIC implementation of our novel algorithm and architecture will perform by one or two orders of magnitude faster than a software implementation and even the FPGA-implementation. Synthesis results proved the design to be highly space-efficient.

In addition, our novel DOP overlap test algorithm lends itself well to parallelization. Only a slight modification of the controller is necessary to use multiple pipelines to test multiple candidate separating axes in parallel. In conjunction with its low area consumption, this allows for an easy implementation of a highly parallelized architecture with an expected speed-up linear to the number of pipelines.

Here, memory bandwidth becomes the limiting factor for speed of collision queries. Possible solutions could be compression of DOP coefficients and the introduction of a cache.

Another important topic is fixed-point accuracy. Here, a lot of different ways to get smaller projections are conceivable.

Collision detection of deformable objects is another important issue. It remains an open problem, which algorithms and data structures are best suited for hardware implementation. Furthermore, we will evaluate different kinds of primitives like quadrangles and NURBS.

REFERENCES

[1] Ageia. White paper, May 2005. `http://www.ageia.com/pdf/wp_2005_3_physics_gameplay.pdf`.

[2] George Baciu, Wingo Sai-Keung Wong, and Hanqiu Sun. RECODE: an image-based collision de-

tection algorithm. *The Journal of Visualization and Computer Animation*, 10(4):181–192, October - December 1999. ISSN 1049-8907.

[3] Jens Eckstein and Elmar Schömer. Dynamic Collision Detection in Virtual Reality Applications. In *Proc. The 7-th Int'l Conf. in Central Europe on Comp. Graphics, Vis. and Interactive Digital Media '99 (WSCG'99)*, pages 71–78, Plzen, Czech Republic, February 1999. University of West Bohemia.

[4] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In *SIGGRAPH 96 Conference Proceedings*, Holly Rushmeier, Ed., pages 171–180. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[5] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware. In *Graphics Hardware 2003*, pages 25–32, July 2003.

[6] Alexander Gress and Gabriel Zachmann. Object-Space Interference Detection on Programmable Graphics Hardware. In *SIAM Conf. on Geometric Design and Computing*, M. L. Lucian and M. Neamtu, Eds., pages 311–328, Seattle, Washington, November13–17 2003. Nashboro Press.

[7] Stefan Hochgürtel, Andreas Raabe, Gabriel Zachmann, and Joachim K. Anlauf. Collision Detection for k-DOPs using SAT with Error Bounded Fixed-Point Arithmetic. Tech. rep., University of Bonn, September 2005. `http://www.collisionchip.de`.

[8] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, September 1995. ISSN 1077-2626.

[9] Jan Klein and Gabriel Zachmann. The Expected Running Time of Hierarchical Collision Detection. In *SIGGRAPH 2005, Poster*, Los Angeles, August 2005. `http://www.gabrielzachmann.org/`.

[10] Dave Knott and Dinesh K. Pai. CInDeR: Collision and Interference Detection in Real-Time Using Graphics Hardware. In *Proc. of Graphics Interface*, Halifax, Nova Scotia,Canada, June11–13 2003.

[11] Karol Myszkowski, Oleg G. Okunev, and Tosiyasu L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995. ISSN 0178-2789.

[12] Andreas Raabe, Blazej Bartyzel, Joachim K. Anlauf, and Gabriel Zachmann. Hardware Accelerated Collision Detection — An Architecture and Simulation Results. In *Design Automation and Test (DATE)*, Munich, Germany, March7–11 2005. `http://www.gabrielzachmann.org/`.

[13] Andreas Raabe, Blazej Bartyzel, Joachim K. Anlauf, and Gabriel Zachmann. Hardware Accelerated Collision Detection — An Architecture and Simulation Results. In *Design Automation and Test (DATE)*, Munich, Germany, March7–11 2005. `http://www.collisionchip.de`.

[14] Gino Johannes Apolonia van den Bergen. *Collision Detection in Interactive 3D Computer Animation*. PhD dissertation, Eindhoven University of Technology, 1999.

[15] Gabriel Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, pages 90–97, Atlanta, Georgia, March 1998.

[16] Gabriel Zachmann and Günter Knittel. An Architecture for Hierarchical Collision Detection. In *Journal of WSCG '2003*, pages 149–156, University of West Bohemia, Plzen, Czech Republic, February3–7 2003. `http://www.gabrielzachmann.org/`.

[17] Gabriel Zachmann and Günter Knittel. High-Performance Collision Detection Hardware. Tech. Rep. CG-2003-3, University Bonn, Informatikk II, Bonn, Germany, August 2003. `http://www.gabrielzachmann.org/`.

Making Grass and Fur Move

Sven Banisch and Charles A. Wüthrich
CoGVis/MMC – Faculty of Media
Bauhaus-University Weimar
D-99421 Weimar (GERMANY)
E-Mail: [sven.banisch|caw]@medien.uni-weimar.de

ABSTRACT

This paper introduces physical laws into the real–time animation of fur and grass. The main idea to achieve this, is to combine shell-based rendering with a mass-spring system. In a preprocessing step, a volume array is filled with the structure of fur and grass by a method based on exponential functions. The volumetric data is used to generate a series of two dimensional, semitransparent textures that encode the presence of hair or of the blades. In order to render the fur volume in real–time, these shell textures are applied to a series of layers extruded above the initial surface. Moving fur can be achieved by horizontally displacing these shell layers at runtime through a mass–spring mesh. Four different mass–spring topologies – different arrangements of masses and springs over the grass–covered surface – are introduced and used for animation. Two of them allow the shell layers to separate laterally, so that the "parting" of grass can be simulated. Performance observations prove mass-spring systems to be well-suited for the real–time simulation of fur and grass dynamics.

Keywords

Computer Graphics, Real–Time Animation, Physically–Based Simulation, Mass–Spring Systems

1 INTRODUCTION

Fur and grass are natural materials the real–time *rendering* of which has attracted many computer graphics researchers in the past years. High performance techniques for their fast rendering have been developed. A realistic animation, however, also requires an appropriate simulation of the dynamic behavior of the material. There are rare attempts focussing on this problem (e.g., [23, 14, 7]), but they do not yet use an adequate physical description, and therefore, do not yet provide a satisfactory solution. The motivation of this paper is to develop such a *physically–based* approach by introducing Newtons laws of motion into the real–time animation of fur and grass.

The most common method to render fur– and grass–covered surfaces in real–time is to represent the volumetric structure as a series of layers which are extruded above the initial surface and textured with semi–transparent hair textures. This method (called *shell–based rendering*) was introduced by Lengyel in 2000 [17] and was subsequently improved during the following years [16, 10, 7, 26]. A dynamical simulation of the surface structure can be achieved if the shell layers are horizontally displaced under the influence of external forces [16].

The main issue of this paper is to analyze if, and in what way, *mass–spring systems* can be used to determine the shell layer displacement, hence, if such systems can form a suitable physical model for the real–time simulation of fur and grass dynamics.

A mass–spring system is a set of mass points which are linked by springs. They are usually arranged in a fixed topology. Mass–spring systems are mainly used for the real–time simulation of deformable objects, in particular, they are often involved into the simulation of cloth–like objects [3, 21, 8, 13, 29]. To research the applicability of mass–spring systems in the animation of fur and grass a topology of masses and springs (called *mass–spring topology*) is generated over the fur–covered surface represented by Lengyel's method. The shell method and the mass–spring system are combined to simulate movement.

The remainder of this paper is organized as follows: Section 2 presents a selection of previous related work. Section 3 describes basic algorithms involved into the dynamic grass simulation, and Section 4 then shows how these algorithms have been combined. The simulation results achieved by this combination are presented in Section 5. Finally, Section 6 sums up this work and points out issues for future development.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

2 PREVIOUS WORK

Our approach combines two techniques which are well–developed and have been frequently used in the last years: the shell method as a real–time rendering technique for furry surfaces, and the mass–spring system used for the real–time simulation of deformations. In the following, very briefly, both techniques are separately considered.

Fur and Grass in Computer Graphics

For more than 20 years the image synthesis of grass has been an attractive topic of computer graphics. An early proposal by Reeves and Blau was based on particle systems. Particles were used to create the structure of grass as well as to render it [24]. Following the idea of representing hair as a set of particle traces, Kajiya and Kay introduced the texel – a three dimensional array that stores the hair data – and arranged these texels over an initial surface which is then rendered via ray tracing [12]. With both proposals high–quality image of grass and fur could be achieved at rendering times far beyond real–time.

In 1998, Meyer and Neyret [20] introduced interactive three dimensional textures using a method called slicing. This technique could be adapted to the more specific case of fur in 2000 by Lengyel [17]. His work proposed *shell-based rendering* for the animation of fur and grass in real–time. The shell method represents a furry surface as a series of concentrically ordered shell layers. The structure of fur and grass is encoded into the alpha channel of the shell textures, which are then applied one by one to their respective shell layers. Since 2000, there have been various authors (e.g., [16, 10, 7, 26]) addressing a more realistic real–time animation of fur and grass, basically, by improving the visual quality and the flexibility of shell-based rendering.

To the authors knowledge, there are only two attempts to a dynamic simulation of fur and grass displayed by a series of shell layers. The first one is a method developed for ATIs RADEON 8000 graphic card series which uses the programmable graphics hardware for the rendering as well as for the dynamical simulation [14]. The fur–covered object is rendered in two stages: one to compute the shell layer displacement, and a second one to draw the fur. The second idea, proposed in 2003 [7] and extended in 2004 [26], introduces so called ”wind vectors” which, stored at each vertex of the object, determine the shell layer displacement. Global forces resulting from wind, gravity, motion and momentum are considered for the calculation the individual ”wind vectors”. Both attempts do not use the Newtonian laws of motion, and a ”damping force” is added by ”merging” the forces of two consecutive frames.

Mass–Spring Systems

Mass-spring systems are quite frequently used models to approximate the physical behavior of deformable objects (e.g., [22, 3, 8, 5, 15, 13, 21, 11]). They are easy to understand and to implement, highly parallelizable [29], and can achieve real–time rates. The nonrigid object is modeled as a set of mass points and springs in a fixed topology. The forces in between two masses are linearly approximated with Hooke's law and neglected for points that are not connected by a spring. These assumptions diminish the number of computations, so that mass-spring systems can simulate complex deformable objects at interactive rates.

Nevertheless, mass-spring systems do involve the solution of a system of ordinary differential equations (ODEs), because they base on Newtons fundamental law $\vec{F} = m\vec{a}$. In order to solve the equations of motion and to simulate such a system over time, a discretization in time has to be applied. A discrete time step is introduced and used for the numerical time integration of the equations ”governing” the motion. Numerical time integration methods include explicit methods, such as the Runge Kutta method, and implicit predictor-corrector schemes.

The range of problems mass–spring systems have been used to solve is rather wide: from the simulation of cloth–like objects in virtual environments (e.g., [21, 8, 13, 29]), where detailed overviews are available [3, 18], over the simulation of rigid bodies attached to elastic ones [11], to the simulation of hair(e.g., [25, 1, 28, 27]), where the recent technological advances are summarized in [19]. Within this work, ways of adjusting masses and springs to shell–based rendering have been developed, so that the range of application of mass–spring systems is widened.

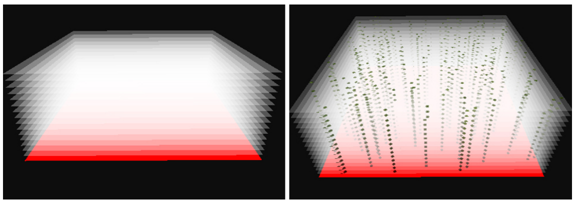
3 BASIC ALGORITHMS

This section briefly discusses the three basic components involved into the grass animation: shell–based rendering, shell texture generation and mass–spring systems.

Shell–Based Rendering

The implementation of the shell method was based on Lengyel's initial proposal [17] as well as on the enhancements proposed in [16]. A number of shell layers (ranging from 12 to 128 in this work) is extruded above the initial surface (”skin”) of the object. This extrusion is determined by the surface normal vectors, specifying the direction, and the *inter–shell distance*, giving the distance between two consecutive layers.

Figure 1 illustrates the arrangement of the shell layers. To make the shell layers represent fur or grass, the corresponding semitransparent shell textures are applied to them. Figure 1b illustrates how this works.



(a) **(b)**

Figure 1: Shell layers are extruded above the initial surface **(a)**, and shell textures are applied to them **(b)**.

The shell textures only encode the structure of grass. Visual attributes, such as color and shadow, have to be added to obtain appealing renderings. In this work, per pixel lighting on programmable graphics hardware has been used to gain a higher flexibility in combining colors and evaluating different lighting models. The color is determined by the color of the initial surface, either by a "skin" texture or by per vertex colors. The shadow which individual blades throw onto each other (self–shadowing) is approximated by darkening the shell layers the closer they are to the initial surface[1]. Finally, a diffuse shading model, based on Lambert's cosine law, is applied to the furry surface. Figure 2 illustrates how adequate renderings of the furry dog are achieved by adding color, self–shadow and shading to the structure represented by concentric shell layers.

Shell Texture Generation

The generation of shell textures is of vital importance to the visual result of the shell method. The textures should encode the structure of fur and grass in a realistic fashion. The structure of fur and grass is rather simple: a large number of distinct individual elements is stochastically distributed over a surface. Most frequently, particle systems have been used to generate these individual elements (e.g., [24, 12, 17, 16, 10]).

There is, however, another generating method, which uses exponential functions to define the shape of the individual blade [4]. A large number of these curves is distributed within a volume data set which is then used to generate the shell textures by horizontally slicing it. To make the structure inhomogeneous, so to say more realistic, the stochastic parameters *length*, *direction of inclination* and *bending* are introduced and applied to each individual. This simple method proved to be well–suited for the generation of grass–like structures.

Following an idea thought by Kajiya and Kay in 1989 [12], we additionally generate an "undercoat" – a dense coat of short hair – in order to model an accurate structure of fur as well.

[1]The most distinctive characteristic of shadow within grass and fur is that it increases the further one approaches the ground. The method proposed in [2] is based on this characteristic.

Mass–Spring System

A mass-spring system is a fixed topology of mass points which are connected by springs. The animation of a system of n mass points requires the solution of the system of n second order ODEs

$$M\ddot{p} = F(t, p, \dot{p}). \quad (1)$$

The $n \times n$ dimensional matrix M stores the masses of all mass points on its diagonal, the n dimensional vector $p = (p_1, p_2, \ldots, p_i, \ldots, p_n)^T$ represents the positions of all points, and F is a function which describes the forces on the system at the time t. This system of n second order ODEs can be reduced to the system of $2n$ ODEs of first order

$$\frac{d}{dt}\begin{pmatrix} p \\ \dot{p} \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} p \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1}F(t, p, v) \end{pmatrix} \quad (2)$$

by substituting $v = \dot{p}$, where v is a n dimensional vector containing the velocities of all mass points.

There are several different methods to numerically solve such a system of ODEs. Explicit methods are very fast at the expense of accuracy and possible instabilities. Implicit methods are considered more stable, but are computationally more expensive. Since it was not clear how fast the mass–spring system has to be for the physical simulation of fur and grass, and how stable it behaves, both an explicit and an implicit *mass–spring solver* have been implemented. A detailed description of both solvers will not be presented within this paper. We refer the reader to standard literature on ODE solvers (e.g., [6, 3, 9]).

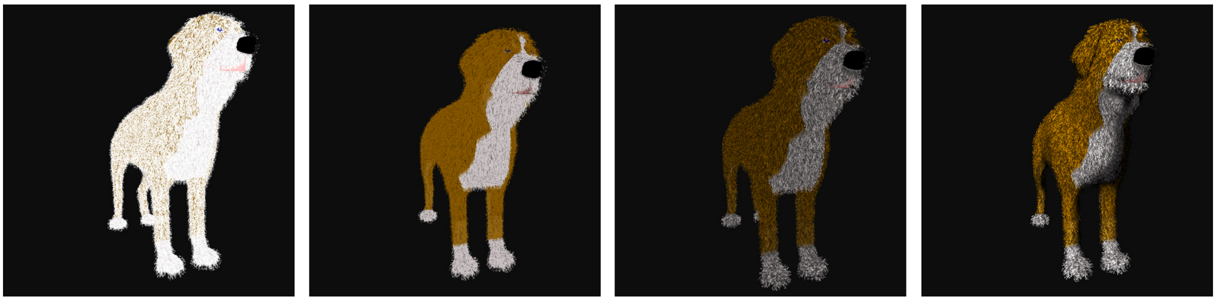
4 COMBINING THE METHODS

Grass represented by a series of shell layers moves if the shell layers are horizontally displaced [16]. All the previously proposed methods for moving grass animation use this characteristic [14, 7, 26]. The methods differ in the way the shell layer displacement is determined, hence in the physical laws modeling the grass movement.

In this work, mass–spring systems form this physical model. In order to combine such a system with Lengyel's rendering method, masses and springs are generated over the surface and attached to the shell layers. Different arrangements of masses and springs are called *mass–spring topologies*, and will be described first. After that, the method by which the movement of mass points is transformed into shell layer movement will be described.

Mass–Spring Topologies

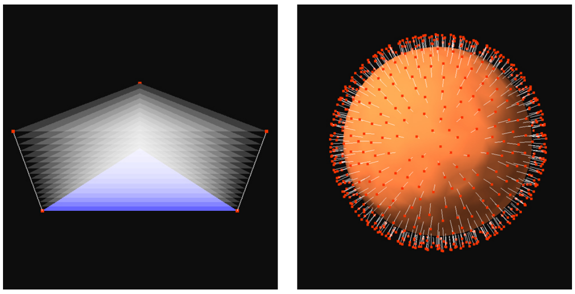
The first and most simple mass–spring topology is called *spring–stick topology*. At each vertex of the initial surface two mass points are generated – the first one attached to the initial surface by setting its mass



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

Figure 2: Creating the image of fur by adding color **(b)**, self–shadowing **(c)** and a diffuse shading **(d)** to the fur structure **(a)**.

sufficiently large, and the second one connected to the uppermost shell layer. Both points are connected by a single spring. In Figure 3, the results of the spring–stick topology generated over an entire surface are illustrated.



(a) **(b)**

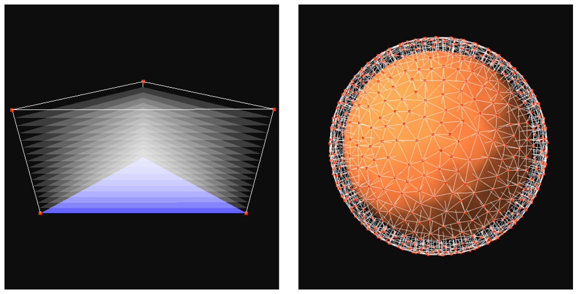
Figure 3: Spring–sticks applied to a single triangle **(a)** and to a sphere **(b)**.

The second topology is called *prism topology* since the masses and the springs form a prism on each surface triangle. Mass points are generated in the same way as it was done for the spring–stick topology. Also corresponding to the spring–sticks, the mass points are vertically linked by a spring. Additionally, we connect the mass points on the uppermost shell layer by generating springs along the edges of the uppermost layer. The resulting prisms are shown in Figure 4.

An interesting effect in grass animation is the simulation of *parting*. Parting means that fur and grass can be split, and that clusters of hair can move independently from other ones. This effect can be simulated if shell layers separate laterally[2]. The parting of fur and grass has been integrated into this work by implementing two additional topologies – *separable spring–sticks* and *separable prisms*.

Both separable mass–spring topologies are build by treating the triangles separately for the generation of mass points and springs. The triangles of the initial surface are handled one by one to attach a mass point

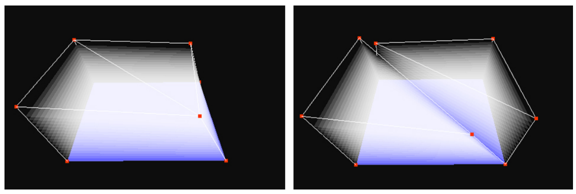
[2]In [16], Lengyel et al. already saw this possibility.



(a) **(b)**

Figure 4: Prisms applied to a single triangle **(a)** and to a sphere **(b)**.

to each of its vertices. Then, mass points are assigned to the vertices of the corresponding uppermost shell layer triangle. The way springs are generated depends on the respective mass-spring topology. In fact, we treat every triangle as being a distinct surface, as a result that the number of mass points and springs needed for an entire surface is increased using these topologies. The separable topologies, however, allow the simulation of parting since neighboring shell layer triangles are not connected. This is shown in Figure 5b.



(a) **(b)**

Figure 5: Neighboring shell layer triangles are connected **(a)**. The shell layers separate laterally **(b)**.

Of course, there are further possibilities of arranging masses and springs over a surface. More springs could be added to connect masses of the initial surface and masses of the uppermost shell layer. Also, mass points could be attached to a larger number of layers. This paper, however, only considers the

 ISBN 1213-6972 ISBN 80-86943-09-7

four mass–spring topologies spring–sticks, separable spring–sticks, prisms and separable prisms which have been introduced above.

The mass-spring topologies are generated in a preprocessing step. During the real time simulation, external forces are calculated and applied to the respective mass points. Then, the mass points positions are updated by the mass-spring solver. In the current state of the application, the external forces are with respect to gravity, wind and motion of the object. Additionally, it is possible to locally apply forces by user input.

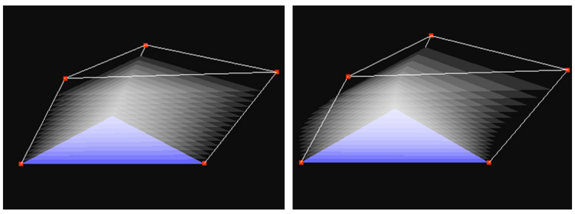
Shell Layer Displacement

Masses are attached to the initial surface and to the uppermost shell layer only. For this reason, a method to determine the shell layer displacement of layers which are not connected to a mass point is necessary. Such a method should approximate the natural bending of the blades. Bakay [7] proposes a technique which is based on trigonometric functions. However, this method is computationally expensive, and a new method has been developed.

During runtime, for every vertex, we first compute the *scale vector*

$$\vec{s}_{i,j} = \frac{p_j - p_i}{|p_j - p_i|}, \tag{3}$$

which is defined by the positions of its two mass points p_i, p_j. Since fur is usually rendered at large scales, it will often be sufficient to linearly displace the shell layers along this vector as shown in Figure 6a.



(a) **(b)**

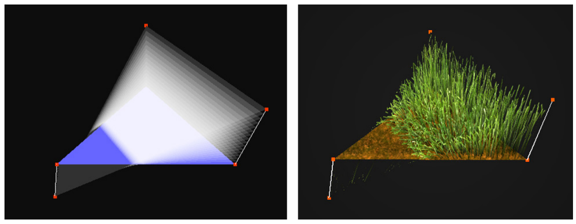
Figure 6: Linear displacement **(a)** and the approximation of the natural bending **(b)**.

In order to obtain an approximation matching better natural bending, the shell layer displacement is composed of a horizontal part $\vec{h}(l_i)$ and a vertical one along the vertex normal vector $\vec{n}$. The quantity of vertical displacement is determined by the fixed inter-shell distance. The horizontal displacement is defined by

$$\vec{h}(l_i) = \left(\frac{i}{n_{layers}}\right)^k (\vec{s} - \vec{n}), \tag{4}$$

which increases with the height of the layer l_i. Figure 6b shows how the shell layers separate from the straight line defined by the vertical springs. Note that the mass points, initially created on the uppermost layer, separate from it. This is because of the vertical displacement determined by the fixed value.

With combining the mass-spring system and the shell method as described, there is one general difficulty: once an external force has been applied to a mass point, it is free to move and does not return to its initial position. Moreover, it is possible that mass points move to the other side of the initial surface (inside the object). Figure 7 illustrates that fur flips to the wrong side of the surface if the mass points move accordingly.



(a) **(b)**

Figure 7: Grass flips to the wrong side of the surface.

In order to solve this problem, we use the normal vector $\vec{n}$ and the vector $\vec{s}$ to compute forces by which the mass points on the uppermost layer are forced to return to their initial position. The force $\vec{F}$ applied to the respective mass point is calculated by

$$\vec{F} = (\vec{n} - \vec{s})(1 - (\vec{n} \cdot \vec{s}))w, \tag{5}$$

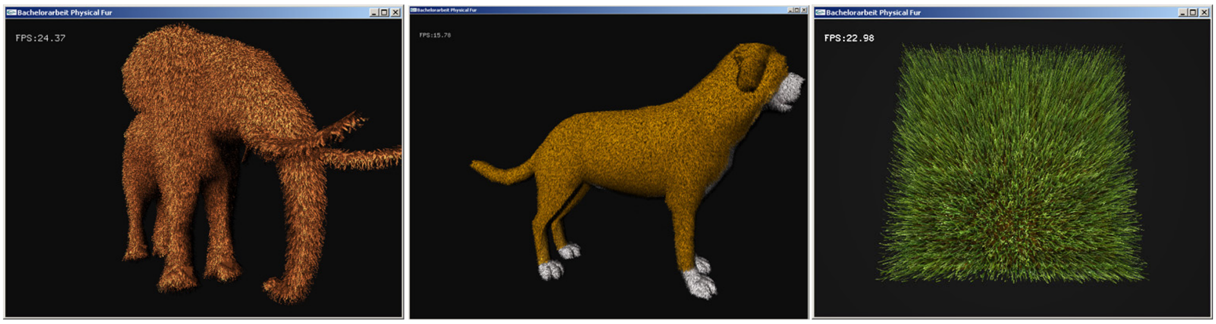
where the term $(\vec{n} - \vec{s})$ determines the direction of the force. The term $(1 - (\vec{n} \cdot \vec{s}))$ is zero in rest state, and increases with the angle formed by $\vec{n}$ and $\vec{s}$. Additionally, an adjustable weight w determines the magnitude of the force. With this procedure, the mass–spring topologies always return to their initial state, since the mass points on the uppermost layer are permanently forced to return to their initial position. Grass does therefore not flip to the wrong side of the surface.

5 RESULTS

Performance Analysis

The moving grass animation presented in this paper was developed and tested on a medium level platform (P4 with 1,7 GHz and 512 MB RAM, Nvidia GeForce 5700 FX with 256 MB). Benchmarks have been done for the three different models shown in Figure 8. Four topologies (spring-sticks, separable spring-sticks, prisms and separable prisms) are compared with each other regarding to the update times. The analysis also takes into account how much of the total time is needed to render the models. Moreover, the two types of mass–spring solver the explicit and the implicit variant are considered.

The first model is an elephant, consisting of 623 vertices and 1148 faces. It is rendered with 16 shell layers, a screen size of 640 $\times$ 480 and a screen coverage



(a) **(b)** **(c)**

Figure 8: An elephant **(a)**, a dog **(b)** and a square of grass **(c)** are used for the benchmarks.

(influencing the rendering speed since per pixel lighting is used) as shown in Figure 8a. The resolution of the shell textures, which is an interesting performance feature if many shell layers are used, is 128 × 128.

The size of the physical system depends on the mass–spring topology. 1247 mass points and 623 springs (one spring for each vertex) are generated if spring–sticks are used. In the largest case, using separable prisms, 6888 mass point and 6888 springs are generated and have to be updated during every frame.

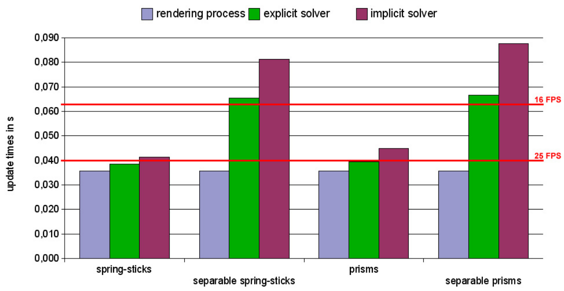


Figure 9: Update times of the elephant model.

The update times of the elephant model, shown in Figure 9, prove that the method is fast enough for real–time use. Spring–sticks and prisms work at update rates of 22 to 27 frames per second. Even when using the potentially slower implicit solution method, no significant slowdown can be noticed. Also, the separable topologies simulated using the explicit solver run at update rates which are still interactive. On this account, the proposed technique is faster than the method proposed in [26], even though the underlying physical laws are of higher complexity.

The dog model consists of 1872 vertices and 3220 faces, and is rendered using 12 shell layers. The size of the screen is 1024 × 768 and the resolution of the shell textures 128 × 128. The screen coverage is shown in Figure 8b. The benchmarking results of the second model are shown below.

The performance evaluation of the dog shows that the rendering process alone is near the limits of inter-

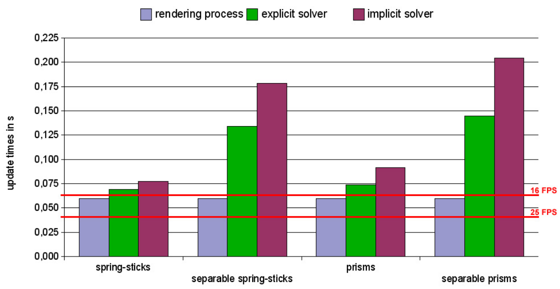


Figure 10: Update times of the dog model.

active application (≈ 17 FPS). Using the spring-sticks for the dynamical simulation does not noticeably slow down the system, and the interactive moving fur simulation of a model of more than 3000 faces and 1800 vertices is feasible, provided that the number of shell layers is reduced to 12.

The last model considered here is a grass model consisting of 13 vertices and 16 faces. It is rendered with 64 shell layers, a texture resolution of 64 × 64 and a screen size of 640 × 480. The amount the object covers of the screen is shown in Figure 8c. It is a crucial performance feature for this example since many shell layers are used to represent the grass.

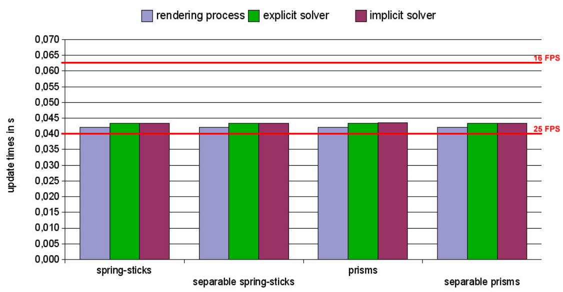


Figure 11: Update times of the grass model.

We can see only slight differences in the performance results. Nearly all the time needed to process the animation is required by the rendering, not by the physical system. Even the more complex mass-spring topologies, simulated by the computationally more expensive implicit mass-spring solver, can be used for the physical simulation of long grass without causing the system to slow down.

The performance analysis shows that the real bottleneck of the system is the shell method, and not the physical simulation. This is obvious for the grass example, since the size of the mass-spring system is very small. But also for the moving fur simulation of the dog and the elephant, using non-separable topologies, the computational power needed for rendering the shell layers takes the greater part of the total performance.

Animation Quality

The usage of a mass-spring system in grass simulation allows a wide range of different effects. Such a system can react to arbitrarily applied forces, and realistic animation effects can be achieved if the forces that act on the system approximate the forces acting in the real world. Mass-spring systems, moreover, provide the possibility to control the dynamical behavior of fur by adapting the physical values which determine how the system behaves.

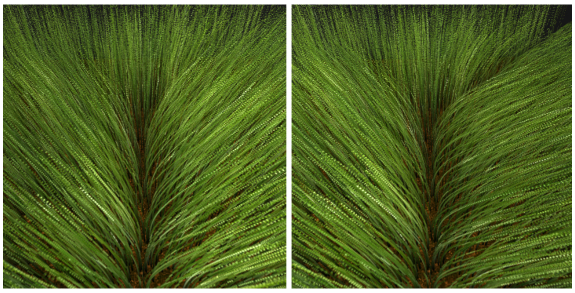


Figure 12: A close view on the parting of grass.

Unfortunately, it is rather difficult to present animation results in writing. Therefore, this paper is accompanied by a video, which shows furry objects in captured animation. Furthermore, in order to provide a rough idea of the quality of animation, Figure 13 shows a sequence of grass waving in the wind. Figure 12 presents a close view on the parting of grass, and gives an idea how this feature can enhance the realism of a grass animation.

6 CONCLUSIONS

A new method for the simulation of grass dynamics has been developed by combining the shell method and mass–spring systems, so that the range of application of mass–spring systems has been widened. Performance observations have proven the methods applicability of being used in a real–time context.

To conclude: mass–spring systems are well–suited to simulate dynamical effects of grass and fur.

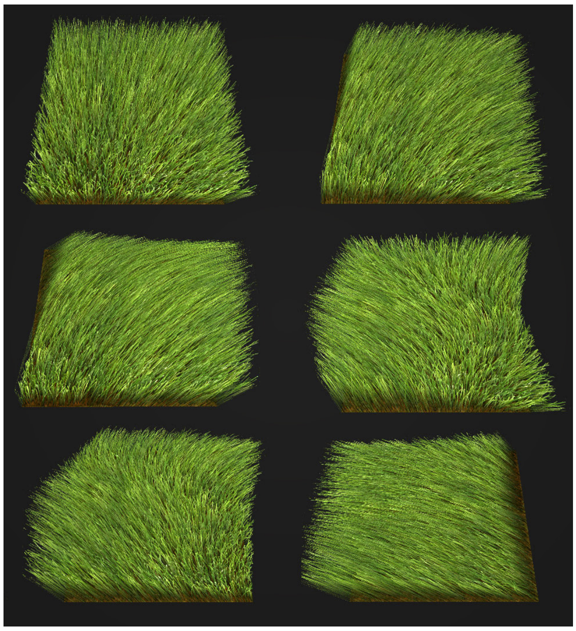


Figure 13: A sequence of grass waving in the wind.

Nevertheless there are several issues that could be addressed by future work.

First and foremost, it will be fruitful put some effort in optimizing the implementation, and ways to fully exploit all capabilities of new graphic cards hardware should be discussed. We believe that this can speed up the application greatly.

An important question to be answered concerns the dynamical effects resulting from motion of the object. There will be no need to calculate any external force, if motional changes of the object are directly transformed into motional changes of the mass points on the skin. Masses on the uppermost shell would react automatically. It is possible that there will be effects on stability of the mass-spring system. However, this way of handling motional changes of the object has to be implemented in future, since a lot of improvement to the moving fur animation is to be expected.

With the development of new graphics hardware optimized for ray tracing, it might also be possible in future, that fur and grass can be interactively rendered by three dimensional textures, and that the texel approach [12] will be taken back in consideration. Visual quality of fur and grass would be improved by great amounts. The proposed method can be adopted to simulate the physical behavior of structures represented by three dimensional textures.

Last but not least, it might be interesting to research more mass–spring topologies.

All in all, the presented technique yields good results and will be used for the simulation of fur and grass in future real–time productions, such as computer games or virtual puppetry.

7 ACKNOWLEDGMENTS

The authors wish to thank all participants of the project "puppets & hands". In particular, we thank Tobias Hofmann, Benjamin Schmidt, Uwe Hahne and Bernhard Bittdorf for their support.

References

[1] K. Anjyo, Y. Usami, and T. Kurihara. A simple method for extracting the natural beauty of hair. In *Proceedings of SIGGRAPH '92*, pages 111–120, 1992.

[2] D. C. Banks. Illumination in diverse codimensions. In *SIGGRAPH '94*, pages 327–334, 1994.

[3] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM Press, 1998.

[4] H. B. Bidasaria. A new method for modeling of hair-grass type textures. In *CSC '95: Proceedings of the 1995 ACM 23rd annual conference on Computer science*, pages 109–113. ACM Press, 1995.

[5] D. Bourguignon and M.-P. Cani. Controlling anisotropy in mass-spring systems. In *Proceedings of the 11th Eurographics Workshop on Computer Animation and Simulation 2000*, Springer Computer Science, pages 113–123. Springer-Verlag, August 2000.

[6] I. Bronstein, K. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, Thun, third edition, 1997.

[7] M. Brook Maurice Bakay. Animating and lighting grass in real-time. Master's thesis, The University Of British Columbia, 2003.

[8] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, June 1999.

[9] M. Hauth. *Visual Simulation of Deformable Models*. PhD thesis, Eberhard–Karls–Universität Tübingen, 2004.

[10] J. Isidoro and J. L. Mitchell. User customizable real–time fur, 2002. SIGGRAPH 2002 Technical Sketch, pp.273.

[11] J. Jansson and J. Vergeest. Combining deformable and rigid body mechanics simulation. *The Visual Computer*, pages 280–289, February 2003.

[12] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 271–280. ACM Press, 1989.

[13] Y.-M. Kang and H.-G. Cho. Complex deformable objects in virtual reality. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 49–56. ACM Press, 2002.

[14] T. Kano. Dynamic fur demo. ATI Developer: Source Code http://www.ati.com/developer/indexsc.html.

[15] U. G. Kühnapfel, H. K. Çakmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24(5):671–682, 2000.

[16] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe. Real-time fur over arbitrary surfaces. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 227–232. ACM Press, 2001.

[17] J. E. Lengyel. Real-time hair. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 243–256. Springer-Verlag, 2000.

[18] N. Magnenat-Thalmann, F. Cordier, M. Keckeisen, S. Kimmerle, R. Klein, and J. Meseth. Simulation of Clothes for Real-time Applications. In *Proceedings of Eurographics 2004, Tutorial 1*, 2004.

[19] N. Magnenat-Thalmann, S. Hadap, and P. Kalra. State of art in hair simulation. *International Workshop on Human Modeling and Animation, Seoul, Korea*, pages 3–9, 2002.

[20] A. Meyer and F. Neyret. Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*, pages 157–168. Eurographics, Springer Wein, July 1998.

[21] M. Meyer, G. Debunne, M. Desbrun, and A. H. Barr. Interactive animation of cloth-like objects in virtual reality. *Journal of Vizualisation and Computer Animation*, 2000.

[22] G. S. P. Miller. The motion dynamics of snakes and worms. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 169–173. ACM Press, 1988.

[23] F. Perbet and M.-P. Cani. Animating prairies in real-time. In *ACM Interactive 3D Graphics*, USA, Mar 2001.

[24] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *SIGGRAPH '85 Comput. Graph.*, 19(3):313–322, 1985.

[25] R. E. Rosenblum, W. E. Carlson, and E. Tripp, III. Simulating the structure and dynamics of human hair: modelling, rendering and animation. 2(4):141–148, Oct.–Dec. 1991.

[26] G. Sheppard. Real–time rendering of fur. Bachelor thesis, The University of Sheffield, 2004.

[27] K. Ward and M. C. Lin. Adaptive grouping and subdivision for simulating hair dynamics. In *Pacific Conference on Computer Graphics and Applications*, pages 234–243, 2003.

[28] X. D. Yang, Z. Xu, J. Yang, and T. Wang. The cluster hair model. *Graphical Models*, 62(2):85–103, 2000.

[29] F. Zara, F. Faure, and J.-M. Vincent. Physical cloth simulation on a pc cluster. In *Parallel Graphics and Visualisation 2002*, 2002.

Crowd Self-Organization, Streaming and Short Path Smoothing

Stylianou Soteris
University of Cyprus
75 Kallipoleos Str.

Cyprus (CY-1678), Nicosia,

ssotos@ucy.ac.cy

Chrysanthou Yiorgos
University of Cyprus
75 Kallipoleos Str.

Cyprus (CY-1678), Nicosia,

yiorgos@ucy.ac.cy

ABSTRACT

Pedestrians implicitly cooperate by forming lanes inside dense crowd in order to facilitate flow and prevent complete passage blocking. Our aim is to re-enforce this self-organization phenomenon in dense crowd for the purpose of virtual crowd animation and navigation simplification. The mechanism of a *flow grid* is introduced to measure flow over an area. The flow grid is a perception mechanism of the surrounding area and favors dynamic lane formation (streams). It provides feedback to the navigation algorithm of the avatars, to enable them to choose a route that both meets their goal (wanted direction) and a trajectory that assists in self-organization of the crowd.

A very simplified yet fairly effective navigation method suitable for dense crowds is also presented. It demonstrates that self-organization of the avatars can help in simplifying local navigation. The method produces short distance, intermediate positions ahead in time and, as a post-processing step, smoothes them out before the avatar needs to use them.

Keywords

Crowd Navigation, Behavioral Navigation, Pedestrian Simulation.

1. INTRODUCTION

Reproducing believable crowds in virtual environments has always been a challenge. There are many research topics related to crowds. Such topics include character rendering, character animation, navigation, crowd simulation, collision avoidance, social behaviors and many more.

The focus of this paper is on the collective behavior of pedestrians, otherwise known as self-organization. A dense crowd of pedestrians, all trying to travel to unique and independent directions, creates a highly dynamic, changing environment. The pedestrians have to continuously confront many surrounding avatars whose navigation constantly changes. It is not possible to try to predict each opposing pedestrian's path, as that will probably change soon. Long distance path planning inside a dense crowd is futile. For this reason, the flow grid mechanism is introduced to simplify navigation and enable crowd self organization.

In the next section a summary of related work on navigation is presented. Section 3 presents the work on self-organizing crowd, followed by a description of the mechanisms needed to make this method feasible in Section 4. In Section 5 the results of the method are presented and finally in Section 6 a summary is presented and future work is discussed.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2. RELATED WORK

Crowd navigation deals with the problem of steering an avatar inside a large amount of static and moving obstacles (other avatars). The complexity of finding a suitable path increases as the density of moving avatars increases. There are three different approaches to solving the navigation problem. These are path planning, reactive navigation and behavioral navigation. Also of great interest to navigation is the computationally expensive collision detection and avoidance stage.

Most papers treat avatar navigation as a one person steering and path finding problem. The motion of an avatar from a starting point to a destination is treated as a global navigation problem. Various search algorithms try to find a complete path from one point to another. The most popular path planning algorithms are the Randomized Path Planner [Tsa03a], Probabilistic Road Maps as in [Kam04],[Sun05],[Sun04] and A* based Algorithms and variations such as [Cla87a],[Har68a]. In general path-planning algorithms are not a good option for dense crowd navigation because continuous re-planning would be needed due to the high number of dynamically moving objects. This of course would be inefficient.

Significant work has been carried in reactive navigation. Reactive navigation includes force field methods, rule based methods and XZT space methods. Force fields have been used by [Kam04], [Lam04], [Met03] for collision avoidance and smooth steering, but force fields are expensive to update. Rule based systems such as [Los03], [Tcc02],[Tcc01],[Nie03] have proven to be extremely fast as they can deal with thousands of avatars. The behaviors produced are limited only, by the number of rules applied. The Thalmann's also provide a good overview of previous work for groups in [Mus04]. Their work on emergent crowds is described. They model complex behaviors through a combination of attributes and rules, and through finite state machines. Their methods lack a global vision mechanism, thus forcing them to consider each neighboring avatar individually when making a decision. Our method will expand the rule based approach and add collective behavior by creating the mechanisms to read the collective behavior of other avatars in an area. XZT space methods exploit a space-time representation to better represent the movements of dynamic objects in space and time. The added T dimension allows for more accurate planning since it provides knowledge of the positions of all avatars ahead of time. Only a few experiments have been conducted with this method in [Feu00]. The same XZT space approach is also used by [Tsa03] in path planning for groups.

In behavioral navigation researchers try to simulate aspects of pedestrian behavior, mainly grouping, staying together, path following, yielding etc. Flocking [Rey87] is one such example. More recent work on behavioral navigation, include behavior maps [Los03], [Tcc01a], which cause pedestrians to adopt specific behavior for designated areas, according to the behavior map. Complete Sociological models have been proposed by [Sul02] and [Vil03]. Work on groups has also been carried out by [Kam04] and [Tsa03] , but it focuses on path finding and computation optimization for groups rather than behavioral aspects of group.

In crowd simulation and safety, [Sti00] presents extensive research with his thesis. Different behaviors of pedestrians in crowded areas are studied in detail. [Daa03] also reports on similar self-organization and other phenomena.

Collision detection/avoidance (CD) is one of the most expensive operations during navigation. Collision detection methods include bounding volume hierarchies, force fields, simple geometric tests, triangulation methods, proximity queries, occupancy maps, image-space methods and stochastic collision detection. A comprehensive review of CD methods can be found in the tutorial [Tes05].

3. SELF-ORGANIZING PEDESTRIANS

General Concept

Pedestrians follow other people when they travel in the same direction and need effort to overtake them. It is much easier to slow down and follow rather than walk into the opposing pedestrian stream when densely populated. Pedestrians need to be aware of any lane formation phenomena around them if they are to use the lanes to their benefit. Large coherent groups of people are very rare. Pedestrians form informal groups dynamically as they walk. They follow a bunch of people going in the same direction as them to avoid opposing streams [Mus04], [Daa03].

Overall Description of methodology

Initially a *flow grid* is constructed over the walk area. The flow grid measures the densities of pedestrians and their velocities at various directions. Thus the flow grid enables us to detect pedestrian density concentration and dominating direction of flow at any point in the walk area. The avatars use the flow grid information during navigation to select nearby areas that lie towards their final destination and have smaller opposing flow. In essence they avoid areas of high density or high opposing flow, thus lane formation takes place. The avatars continue to select nearby areas until they reach their final destination.

A high level path planning and areas/portals system is assumed on top of our system for navigating through a city. It is also assume that the avatars move on the XZ plane.

Measuring the Flows

Each avatar registers his position and velocity on the flow grid as soon as he moves to a new position. The velocities are separated into X and Z axis components. Positive and negative axis velocities are stored separately, Thus four velocity values are stored at each point, (+vx, -vx, +vz, -vz). Velocities are stored this way so that opposing velocities will not be canceled out.

The avatar's velocity and presence are distributed to the four neighboring grid points as shown in figure 1. The amount distributed to each corner depends on the avatars distance from that corner. The entire avatar density (1.0) is allocated to a corner when the avatar is exactly at that corner and is smoothly interpolated to zero as he moves away to the next corner.

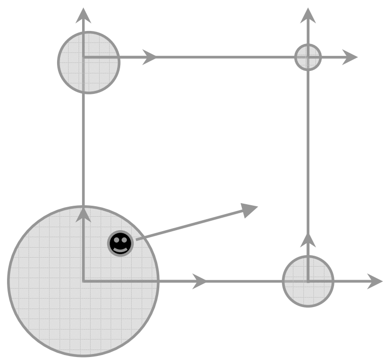


Figure 1. Each avatar is registered on the grid by distributing his density and velocity to the 4 neighboring points

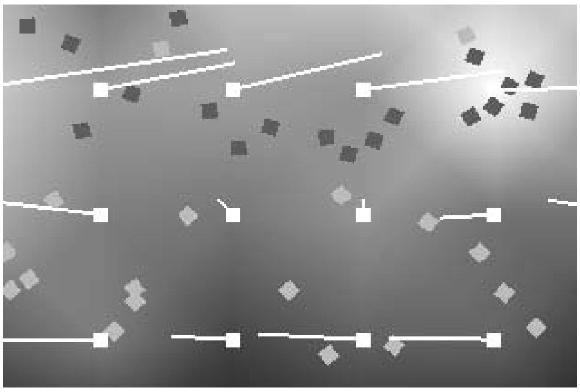


Figure 2. Light intensity shows the density distribution. White lines show the dominating direction of flow at each point. Dark grey avatars heading for the right border and light grey avatars for the left border.

Once all the avatars have registered on the grid, a complete picture of the densities and flows over the entire area is obtained, as shown in Figure 2. Higher intensity grey areas indicate more people near that grid location.

Later on, the flow grid is used to extract information for navigation purposes. Densities and velocities are interpolated between grid positions when information is needed at any in-between point.

Using the Flow Grid to Navigate

A higher layer that assigns long distance destinations to each avatar has been assumed, e.g. [Sty04]. Then the steering towards that destination must be performed. For example an avatar is told to steer from one end of a road or square to another end by the higher layer. The avatar has to navigate to that destination. He does so by selecting intermediate local targets just a few meters ahead of him. By consulting the flow grid at regular intervals the avatar chooses to head for the area with smaller density and smaller opposing flow. To help the avatar decide which area is best, a special weight formula has been constructed. This formula is explained in the box below:

Weight = (1 + D) * (1 + AngleDiff (T, F))

D = density at spot

T = vector showing direction towards target pos

F = vector showing direction of flow at spot

AngleDiff = Angle difference between 2 vectors in radians

The weight is a product of the density at that spot and the angle difference between the direction towards the avatar's target and the dominating direction of flow on the flow grid.

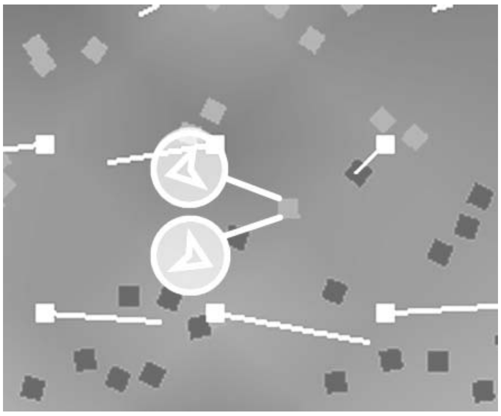


Figure 3. The arrows in circle demonstrate the dominating flow on the *flow grid* at that position.

Two weights are calculated each time the avatar needs to find a new intermediate destination. Each weight lies approximately 2 meters ahead of each pedestrian and to his left and right. The spot with the lowest weight is chosen as a temporary local target as shown in Figure 3.

The weight has been formulated empirically and it gives a nice measurement of the density and opposing flow at any spot.

The avatar continues to steer to the temporary target until a new temporary target is calculated in a short time interval. The grid takes care of the lane formation behavior of the avatars. The steering and collision avoidance method is presented next.

4. LOCAL STEERING AND SMOOTHING

About steering in densely populated areas

Pedestrians walking in densely populated areas change direction to avoid others all the time. Checking for long free paths is not very useful. Even if there is a free path along a direction, it can very easily be claimed by other pedestrians and soon become unavailable. A simple, but slightly deferent than usual, approach has been used by our system.

Description of steering algorithm

A discrete occupancy map is used for collision avoidance. The avatar maintains a list of six positions at any time. Each position is no more than one occupancy cell away from the previous position. At all times the actual avatar position is interpolated between positions 2 and 3. When the avatar reaches position 3, the first position in the list is discarded and a new position is added to the end of the list. The avatar is looking only one occupancy cell around him for a new position. For this reason, a very sharp turn may be needed. Smoothing is used to minimize sharp turns, as a post processing step. Essentially, the first 4 positions are used for curve interpolation and the last 2 are used for path smoothing

Position interpolation

Interpolation allows us to perform path planning only when needed. Practically path planning occurs approximately 3 times per second. The occupancy map cell size is 33cm and the average avatar speed is 100cm per second, so if avatar positions in the list are to be adjacent, position searching will occur 3 times per second. All intermediate positions are interpolated using a Catmul-Rom curve. This is an excellent optimization since simple interpolation is the only operation performed for most frames and path planning is postponed until it needs to be performed.

Next position search

Only the cells around the avatar are checked. Long free paths are not checked, instead, free cells in front of the avatar are checked. Figure 4 shows the search area when looking for a new position. When the avatar runs into dense crowd, he shouldn’t turn back where he came from, as it would look unnatural. For this reason the search distance is being reduced when the rear of the avatar is being searched.

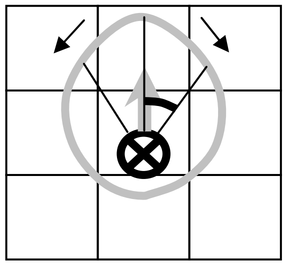


Figure 4. The search area for possible positions is shown here. The search starts from the top and checks left and right at an increasing angle and a reducing distance until an empty cell is found.

Path post-smoothing

Due to the limited search space ahead of the avatar and the densely populated environment, sharp turns can occur quite frequently. For this reason a path smoothing step is performed as post-processing. The curve is smoothed out by moving the in-between position (5th step) to a better location if that is still empty in the occupancy map. Figure 5 demonstrates a smoothing step. Position 5 is moved towards the line formed by positions 4 and 6 in order to minimize the turn angle.

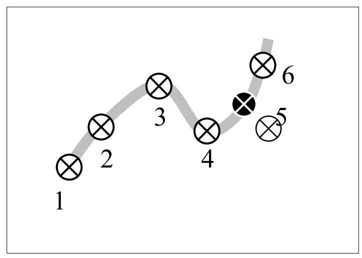


Figure 5. The 6 positions maintained by the avatar. The first 4 are used for curve interpolation. Position 5 is canceled and moved to a better position (black spot) during smoothing.

Each position relates to the occupancy map at the time the position was taken. Because of smoothing, positions taken at past time need to be changed. Occupancy maps for the previous time-steps need to be maintained as well to validate whether such a change is possible. Therefore 2 occupancy maps are maintained, one for the step at position 5 and one for the new position 6. If the new desired position at step 5 is not available on the occupancy map then smoothing does not occur. It would be possible to keep more positions for smoothing, which in turn would require more occupancy maps.

5. RESULTS

Various configurations of crossing and parallel crowd streams have been tested. The results of those tests as well as the local steering results are presented in this section.

Two Parallel But Opposing Streams

The first test was composed of two opposing crowd streams. The avatars are released from random points on one end of the walk area and navigate to another random point on the opposing side of the area. As the two opposing streams meet they start to formulate streams and lane formation becomes more stable after a while (Figure 6). Avatars may need to cross opposing streams if they need to head for a different destination point.

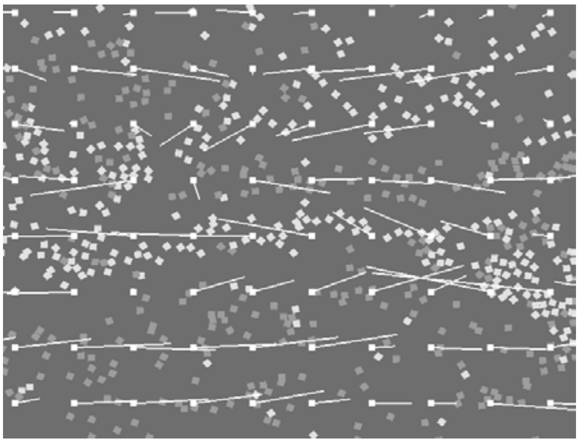


Figure 6. Lane formation between two opposing crowd streams. The dominating direction at each point on the grid superimposed on top.

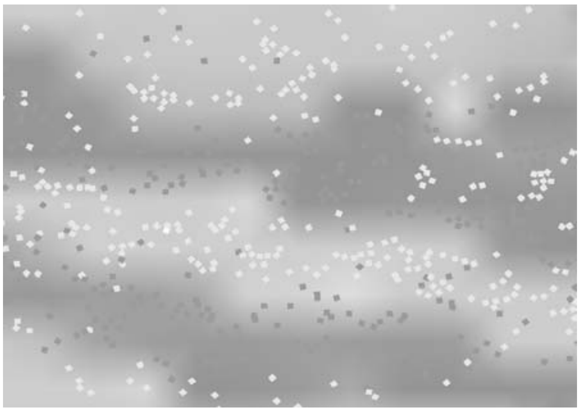


Figure 7. Lane formation between two opposing crowd streams. The flow direction graph is under-imposed.

To better visualize the lanes formulated, a flow direction graph is rendered (Figure 7). The direction of flow at each point on the grid is mapped to a color using a grey color scheme.

Two Crossing Crowd Streams

The results are different for crossing streams. Constant lane formation is not possible, as the two streams need to cross each other. What happens is vertical/horizontal alternation of streams at an area. The momentary lane formation for one direction is followed by alternation once the avatars from the opposing direction start to gather up. The dominating stream blocks the avatars traveling in a vertical direction to the stream, forcing them to slow down. Once the waiting avatars start to gather up, the stream changes direction. Figure 8 shows two crossing streams of avatars.

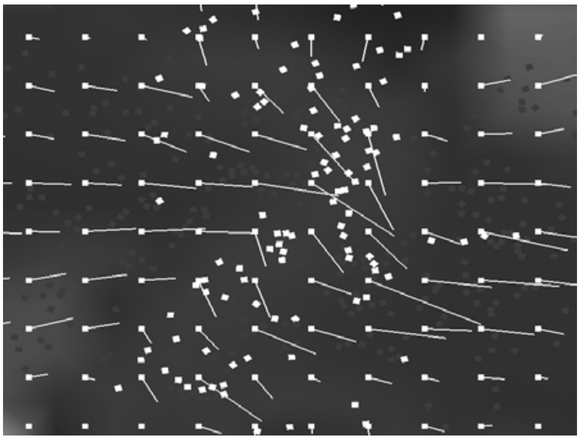


Figure 8. Two crossing crowd streams. Lanes are formed only momentarily, temporarily slowing down avatars that try to cross it.

Figure 9. Two crossing pedestrian streams. The avatars are rendered as impostors. Different colors are used to distinguish avatars belonging to different streams.

It can be seen in Figure 9 that the two crossing streams of avatars mix much more than the previous test case. The avatars in white color need to travel in a perpendicular direction to avatars in grey color. Crossing streams are much harder to distinguish.

Four Crossing Crowd Streams

As Daamen and Hoogendoorn [Daa03] state the resulting self-organization at crossings is just chaos. It has been observed that here too, lane formation is only momentary. It only happens for very short moments, as there are four crowd streams now competing for the same area. The flow measurements on the grid do favor only one direction, the dominating direction at each area. Thus some avatars choose and some avoid an area, which causes the momentary lane formation. Figure 10 demonstrates the resulting flows at a crossing.

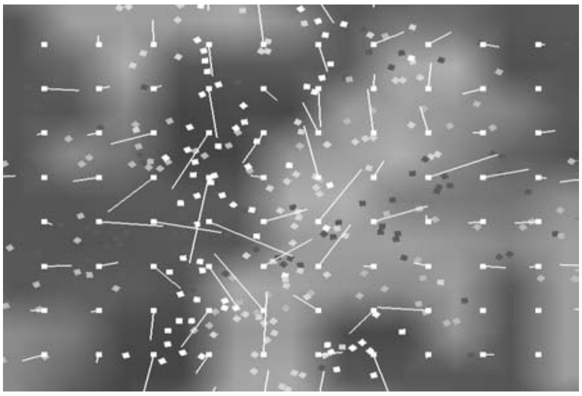


Figure 10. Four crossing crowd streams competing for the same areas. Lanes are formed only momentarily, mostly its only chaos.

Resulting Paths

Some resulting paths are shown in Figures 11 and 12. Figure 11 shows examples of horizontal paths inside two horizontally opposing streams. Figure 12 shows example paths of perpendicularly crossing streams.

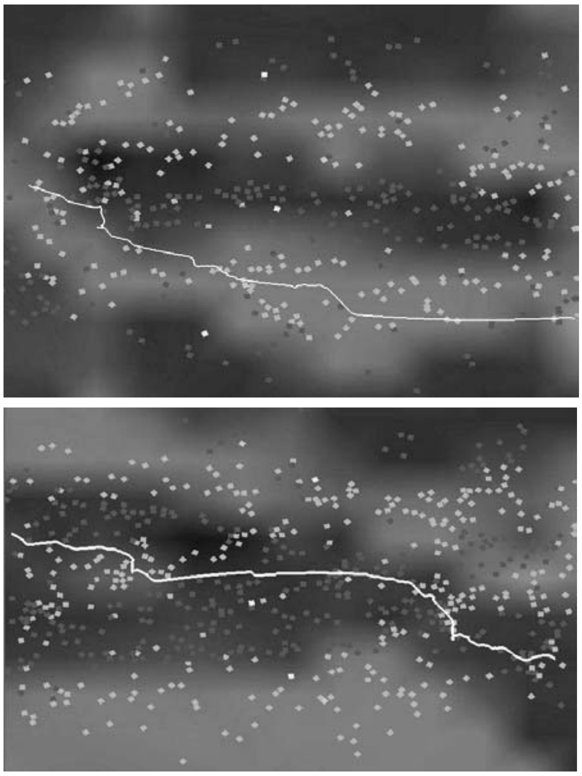


Figure 11. Resulting paths of parallel but opposing streams

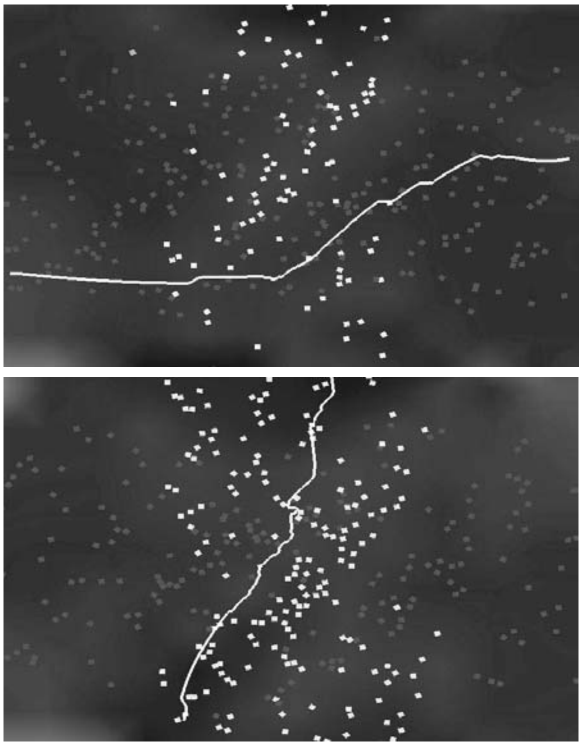


Figure 12. Resulting paths of perpendicularly crossing streams

The granularity of the flow grid affects the lane formation. The lanes become thicker when the grid is coarse, and thinner when the grid is finer. A granularity of 5m to 10m between flow grid points has been found to produce pleasing results.

Performance

The tests have been performed on a Pentium 4, 3Ghz machine with 512Mb RAM and Geforce 5600 with un-optimized code and are running at interactive frame rates for up to 2000 avatars. The simulation area is an open space of approximately 3000 square meters.

The performance cost for the flow grid is minimal. It lies between 1-2% of the navigation performance cost. The relationship of the number of avatars and the flow grid cost is linear. The size of the area and the density of the grid do not affect the performance cost of the flow grid. It is important though to point out that the resulting crowd streams allow for higher densities of avatars to flow through an area, whereas without the flow grid they would jam. Table 1 shows that approximately 20% to 25% more avatars can flow through an area before the crowd jam density limit is reached.

Area in sq.meters	Max. Number of avatars without streaming (approx.)	Max. Number of avatars with streaming (approx.)
3000	~600	~800
5200	~850	~1200
20800	~2500	~3800

Table 1. Improved Density of avatars. Approximate crowd jam limits for different area sizes.

Number of avatars	Average Algorithm Frame Time (msec)	Number of avatars	Average Algorithm Frame Time (msec)
185	11.3	760	48.4
275	16.5	830	55.1
340	21.1	900	59.9
430	26.9	1020	65.4
570	35.2	1230	78.4
670	42.03	1320	86.5

Table 2. Local Navigation performance cost.

The local steering is the most costly part of the algorithm since it includes local collision avoidance. It has been observed that the critical factor in the performance of local steering is the density of the avatars. Table 2 shows the local collision avoidance cost. The performance of navigation is fairly linear (Figure 13) until near maximum capacity is reached. Once the density of avatars becomes too high the cost of collision avoidance rises exponentially due to the high density and continues collisions between avatars. The performance cost for collision avoidance is significantly lower than other methods because each avatar only searches for a step every few frames and only looks at the neighboring cells on the occupancy map around him.

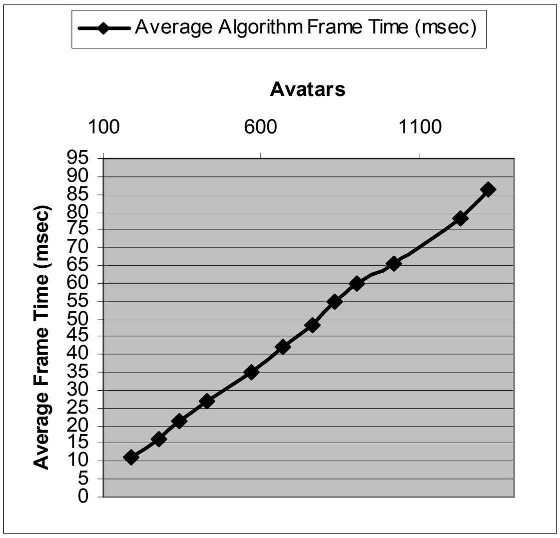


Figure 13. The graph for Table 2. Performance of local steering algorithm.

6. DISCUSION

This paper presented a method for lane formation behavior in crowds and a method for quick steering inside dense crowd. The method has been visually evaluated and it gives pleasing walking crowd results.

The flow grid gives an overall impression of what kind of pedestrian traffic exists in that area. It can easily be used with more complex path planning algorithms for a more detailed path planning approach. The simple path selection algorithm presented here only examines two areas ahead of the avatar, but it has been found to be adequate for dense crowd navigation. Congested areas can also be detected from the grid since they will have high density and low flow.

One important benefit of the steering algorithm is that there is no collision avoidance cost while the avatar positions are being interpolated from one pre-planned step to the next. The collision avoidance costs are thus significantly reduced. Collision avoidance is only performed each time a new step needs to be planned.

The steering method is suitable for a dense pedestrian environment. It cannot operate alone. It needs to be directed towards short distance intermediate waypoints. The intermediate waypoints can be extracted from the flow grid or a similar method. The avatar path list can be extended to include more points and smoothing can be applied to more points in the path list to produce smoother paths.

7. ACKNOWLEDGMENTS

Special thanks to Franco Tecchia for permitting use of his impostor rendering library.

8. REFERENCES

[Cla87] Clarcson, k. (1987). Approximation algorithms for shortest path motion planning. In Proceedings 19th Annu. ACM Symposium of Theory Computation.

[Daa03] Daamen W. and H. S.P. (2003). Experimental Research of Pedestrian Walking Behavior. Louisiana, Louisiana Transport Research Center.

[Feu00] Feurtey, F. (2000). Simulating The Collision Avoidance Behavior of Pedestrians. Dept. of Electronic Engineering. Tokyo, University of Tokyo. MSc: 53.

[Har68] P. E. Hart, N. J. Nilsson, et al. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." IEEE Transactions on Systems Science and Cybernetics(SSC4 (2)): 100-107.

[Kam04] A. Kamphuis and M. H. Overmars (2004). Finding Paths for Coherent Groups using Clearance. Eurographics/ACM SIGGRAPH Symposium on Computer Animation.

[Lam04] LAMARCHE F and D. S (2004). "Crowd of virtual humans: a new approach for real time navigation in complex and structured environments." Computer Graphics Forum 23(3): 509-518.

[Los03] Celine Loscos, David Marchal, et al. (2003). Intuitive Crowd Behaviour in Dense Urban Environments using Local Laws. Proceedings of the Theory and Practice of Computer Graphics 2003, IEEE Computer Society.

[Met03] Ronald, A. M. and K. H. Jessica (2003). Reactive Pedestrian Path Following from Examples. Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003), IEEE Computer Society.

[Mus04] Soraia Raupp Musse, Branislav Ulicny, et al. (2004). Chapter 14 summary of Handbook of Virtual Humans. Groups and Crowd Simulation. D. T. Nadia Magnenat-Thalmann.

[Nie03] C. Niederberger and M. Gross (2003). Hierarchical and Heterogenous Reactive Agents for Real-Time Applications. Eurographics 2003, Granada, Spain.

[Rey87] Reynolds, C. W. (1987). "Flocks, herds, and schools: A distributed behavioral model." In M. C. Stone, editor, Computer Graphics (SIGGRAPH '87 Proceedings), urban walking environments. Journal of Planning Literature: 25-34.

[Sti00] Still, K. (2000). Crowd Dynamics. Department of Mathematics. Warwick, Warwick. PhD.

[Sty04] Stylianou, S, Fyrillas, M and Chrysanthou, Y (2004). "Scalable Pedestrian Simulation for Virtual Cities". ACM VRST 2004, Hong Kong, November 2004.

[Sul02] C. O'Sullivan, J. C., H. Vilhjálmsson, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters, and T. Giang (2002). "Levels of Detail for Crowds and Groups." Computer Graphics Forum, Volume 21 Issue 4: 733.

[Sun04] Sung, M., M. Gleicher, et al. (2004). Scalable behaviors for crowd simulation. EUROGRAPHICS 2004.

[Sun05] Mankyu Sung, Lucas Kovar, et al. (2005). "Fast and accurate goal-directed motion synthesis for crowds". Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005).

[Tcc01] Franco Tecchia, Celine Loscos, et al. (2001). "Agent Behaviour Simulator (ABS): A Platform for Urban Behaviour Development". ACM/EG Games Technology Conference.

[Tcc02] Franco Tecchia, Celine Loscos, et al. (2002). "Visualizing Crowds in Real-Time." Computer Graphics forum Volume 21(Number 4): pages 753-765.

[Tes05] Matthias Teschner, Bruno Heidelberger, et al. (2005). Tutorial: Collision Handling in Dynamic Simulation Environments. Dublin, Ireland, Eurographics 05.

[Tsa03] Tsai-Yen Li and H.-C. Chou (2003). Motion planning for a crowd of robots. International Conference on Robotics and Automation (ICRA). San Diego, CA, IEEE Press.

[Vil03] Marta Becker Villamil, Luiz Paolo Luna de Oliveira, et al. (2003). A Model for Grouping Virtual Individuals Based on Social Behaviors. Intelligent Virtual Agents 2003. Irsee, Germany

Wheelie - Using a Scroll-Wheel Pen in Complex Virtual Environment Applications

M. Wögerbauer
VRVis – Research Center for Virtual Reality and Visualization, Ltd
Donau-City-Strasse 1
A-1220, Vienna, Austria
mwoegerb@vrvis.at

A. L. Fuhrmann
VRVis – Research Center for Virtual Reality and Visualization, Ltd
Donau-City-Strasse 1
A-1220, Vienna, Austria
fuhrmann@vrvis.at

ABSTRACT

Input devices and system control techniques for complex virtual environment (VE) applications are still an open field of research. We propose the use of a scroll-wheel as an extra, dedicated input stream on a tracked stylus for means of system control. We demonstrate how this enhanced stylus can be used together with an appropriate user interface to quickly select commands, change tools, and adjust parameters.

This user interface consists of two different styles: a toolbar and a graphical menu system, both accessible by the same hand that holds the stylus. The scroll-wheel extension does in no way impair the conventional use of the stylus.

Keywords

Input Devices, Scrolling, Virtual environments, Application control, System control, Tool selection, Menu system, Responsive Workbench, 3D Interaction

1. INTRODUCTION AND MOTIVATION

Virtual environments (VEs) promise a great amount of potential as a working environment for applications with a high degree of interactive complexity. They provide a virtual workspace for the user in which his or her hands can be used to grab and manipulate virtual objects in complex workflows. In these workflows, however, we need to make extensive use of overhead tasks like tool switching, adjusting parameters or issuing commands to the system, aside from the main task. Thus, there is a great demand of user interfaces that are capable of performing these so called *system control* tasks efficiently.

In VEs the user interaction can be categorized into four classes of universal interaction tasks [Bow99a].

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

Navigation changes the viewpoint in the environment, and can further be divided into a cognitive part (wayfinding) and a motor part (travel). *Selection* refers to the task of choosing one or more objects from a set which is closely related to the third task of *manipulation*. Manipulation can be described as changing the properties of objects, such as their location in the scene. The above mentioned task of *system control* subsumes all actions that apply commands to change the mode of interaction or the system state. While three-dimensional (3D) navigation and object selection and manipulation were the focus of research in past years, the equally important task of exploring novel system control techniques was left behind. Often application designers settled for using 2D desktop methods implemented in VE applications. In 2D desktop environments the WIMP paradigm (Windows, Icons, Menus and Pointers) is usually adopted as the means of controlling applications. Unfortunately, these interaction techniques cannot be effectively transferred to the 3D world of VEs. 3D interaction methods that employ 2D concepts struggle with the presence of additional degrees-of-freedom (DOF), originating from the extra dimension which hampers the otherwise easy task of selection in menus. When circumventing this problem by bringing 2D into 3D VEs [Lin99a] [Sza97a], i.e. letting the user hold a physical tracked tablet to execute 2D tasks on, we sacrifice the use of one hand to the passive task of holding and gain no additional functionality directly controllable by one hand at the spot of direct manipulation, which would be preferable in complex workflows.

The most widely used input devices on 2D desktop setups are keyboards, computer mice and graphic tablets using a stylus. Their basic concept has remained the same since their advent in the computer world, but even they are still subject to refinement. One important step of enhancing usability of desktop systems was the introduction of the scroll-wheel for computer mice. In document browsing and navigation they take on the otherwise distracting task of scrolling from the main task of reading and editing. The focus of attention does not have to be switched to moving the pointer to a scroll bar, dragging a slider, and then switch back the attention to the document to actually see where we are scrolling. As scroll-wheels have become a de facto standard for mice, it appears to be a logical step for us to equip a stylus, being one of the most frequently used input devices in VEs, with a scroll-wheel as well, and to take advantage of this additional input stream. It will provide application and user interface designers with supplementary input to permit new ways and methods of interaction, not limited to the interaction methods presented in this paper.

We propose a new input device, called *Wheelie*, combined with an appropriate user interface. Wheelie is a multi-stream input device, a tracked stylus with an embedded scroll-wheel that speeds up the execution of many system control tasks found in complex VE applications. It is designed with the purpose to provide a quick way of consecutive selection of different editing tools for direct manipulation or creation of objects, and the possibility of issuing commands extended in a new type of menu system that takes advantage of the constrained input of the scroll-wheel. Wheelie and its user interface are a general purpose approach that can be combined with several other input devices and interaction techniques.

2. RELATED WORK

In spite of the fact that system control tasks account for a large number of interactions, usability of application control in complex VE applications is still mediocre. Research on appropriate interaction techniques and input devices was long neglected and is an open problem. However, much has been done in recent years to mend this situation and some interesting work has emerged.

Kruijff [Kru00a] proposes a categorization of currently used system control methods influenced by the description of non-conventional control techniques by McMillan et al. [McM97a]. It distinguishes between graphical menus, voice commands, gestural interaction, and tools. The group of graphical menus is further divided into hand-oriented menus, converted 2D menus, and 3D widgets. Tools can be divided into physical and virtual tools.

In an attempt to overcome some of the drawbacks of conventional 2D menus transferred to 3D environments and to take advantage of proprioceptive "eyes-off" interaction with the menu, i.e. the person's sense of the position of the body and limbs, Bowman and Wingrave [Bow01a] presented a new menu system called TULIP. It is based on displaying menu items on top of each finger and selecting them by pinching of fingers, using a Pinch Glove™.

Grosjean and Coquillart [Gro01a] developed a novel quick-access menu system for workbench-like VE configurations, called C³ (command and control cube). It is a 3D extension of marking menus, taking advantage of the three dimensions of the space for the selection process. It consists of a 3D grid of small cubes with which commands are associated and executed by positioning a selection pointer in the corresponding cube, using a tracked input device. It is designed with the intent to allow the possibility of rapidly issuing a limited set of commands to the application, similar to hotkeys in 2D desktop environments equipped with a keyboard.

To integrate 2D interaction in 3D VEs, Coquillart and Wesche [Coq99a], and in a similar approach Schmalstieg et al. [Sch99a], proposed the use of transparent props for two-handed application control in projection-based environments. The virtual palette and the PIP (personal interaction panel) consist of a physical tracked transparent plate that is held in the non-dominant hand and can present 2D menus and widgets which are selected by using a tracked pen. The physical surface, acting as a constraint, eases the task of selection and also takes advantage of prop- and body-centered aspects, delivering a kind of passive-haptic feedback.

A way of using the proprioceptive sense of the user is the use of body-centered menus which were explored by Mine et al. [Min97a] and place menu items relative to the user's body. This technique can significantly enhance user performance and even allows for "eyes-off" interaction and selection of menu items and tools. Body-centered menus do not inherently support a hierarchy of menu items and this poses a problem since the selection becomes gradually more difficult with increasing numbers of menu items.

In hand-oriented menus, as used in several applications as a method of system control [Lia94a] [Min97b], the menu items are located on a circular object. In order to select an item the user has to rotate the hand to align the desired menu item in a selection box. The rotation about a single axis, working as a constraint, makes these menus fast and accurate. As is the case with body-centered menus, hand-oriented menus also do not imply a menu hierarchy.

A thorough description of two-handed direct manipulation on the Responsive Workbench was given by Cutler et al. [Cut97a]. It shows how users perform certain tasks in a natural way using both hands. Pinch Gloves and/or a stylus were used as input devices and virtual tools could be selected from a toolbox located in front of the user by either clicking on their representations with the stylus or by pinching them with the gloves. To reduce the number of necessary explicit tool switches, power tools, controlled by the non-dominant hand, and implicit tool transitions were implemented.

The ToolFinger by Wesche [Wes03a] is an interaction technique for VEs that focuses on the problem of frequent tool selection during complex direct manipulation of objects and proposes the integration of the task of tool selection into the workflow of tool application. This is accomplished by subdividing a selection pointer of a tracked stylus into several sections and interpreting an intersection of an object with one of those sections of the pointer as a tool selection. With each section of the ToolFinger a tool is associated. As a result of this method, the ToolFinger is restrained to interactions which are always related to virtual objects. Other interaction modes, e.g. object creation or general application control tasks, are not possible, making the ToolFinger a special purpose technique for direct modification.

In [Ste03a] Stefani et al. discuss requirements for input devices in immersive environments and the design of two such input devices is presented. The Dragonfly and the Bug are two input devices for the dominant and the non-dominant hand, respectively, which work in unison. The Dragonfly is a lightweight, optical tracked, stylus-like pointing device with 6 DOF, not featuring any button. The Bug is, in essence, a 3 DOF (position only) tracked wireless mouse with a jog-dial (or scroll-wheel) and two buttons, held in hand like a TV remote control. The 3 DOF nature of the bug allows the permanent display of a context sensitive, graphical menu and the jog-dial facilitates selection of menu items therein. This way control of the menu system is decoupled from tracking and instead uses the sole input of the jog-dial.

Many interaction techniques described herein are an attempt to overcome the problem of too many degrees-of-freedom of system control interfaces and introduce some sort of constraint in order to ease the task of selection in menus. Wheelie takes the same approach as the Bug when tackling this problem by adding a dedicated one-dimensional discrete input stream to the VE, allowing the user to effectively perform application control tasks without interrupting the current workflow.

3. DESIGN CONCEPT

The basic idea of Wheelie is to extend the input possibilities of a stylus used in many VEs as an input device by adding a scroll-wheel as is commonly used in computer mice. Thus, a separate input stream is available that can, in combination with two buttons, be used to navigate a hierarchical system of application control functionality (Figure 1). Tool selection and navigation in the menu system are mainly controlled by changing between different entries in a set of tools or menu entries using the scroll-wheel.

Several other devices incorporating an additional input stream already exist (e.g. the Wanda available at http://www.wandavr.com/). But most of them are palm held devices that feature a thumb controlled joystick. A joystick is neither as appropriate to perform one-dimensional discrete selection tasks – by virtue of its 2D continuous input stream – as a scroll-wheel, nor do most devices support the use of the extra input stream in conjunction with the other buttons. In this regard Wheelie – as a stylus with a scroll-wheel operated by the index finger and buttons activated by the thumb – offers new ways of interaction previously unavailable.

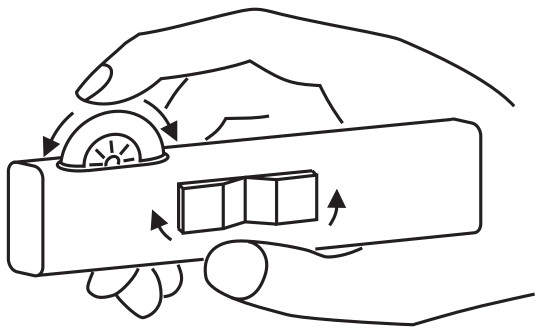


Figure 1: The conceptual design of Wheelie. The index finger is used to operate the scroll-wheel, while the thumb is used to activate two buttons on the side of the stylus.

Generally speaking, system control is the action in which a command is applied to change either the mode of interaction or the system state. In order to issue the command, the user has to select an item from a set and different interaction styles are employed in order to select the commands. As mentioned in the relating work above, these techniques can be put into four categories: Graphical menus, voice commands, gestural interaction and tools. In the course of this work we want to use the following slightly customized terminology of system control tasks: A mode of interaction of direct manipulation or creation of objects is referred to as a *tool*, resulting in *tool switches* when changing between them. Functions of the application that change the state of the system or perform system tasks and should be accessible to the user for execution are called *commands*. If the value of a single variables is subject to explicit changes through the user (*parameterization*), the variable is called a *parameter* and the set of associated valid values is called the corresponding *parameter space*.

As the addition of a scroll-wheel to the computer mouse became a huge success and nearly all mice manufactured today feature one, it is worth looking into the question why it was so widely accepted and what it is used for. The scroll-wheel provides an additional input stream without hindering normal mouse interaction. Predominantly, document scrolling is the domain of the scroll-wheel but other uses have been adopted as well, like picture and document zooming or scrolling through and selecting available options in input fields. Another noteworthy usage of the scroll-wheel is in first-person 3D action games where it is used to quickly swap between different tools of interaction with the environment. It allows the player to change tools while constantly moving at the same time, which is crucial for success in these games.

Besides computer mice, the use of tablets is well established as an input method for graphical art and CAD applications. A currently available input device that probably comes closest in form to the new multi-stream input pen for VE applications proposed in this work is the Wacom Intuos Airbrush-stylus (http://www.wacom-europe.com/uk/products/intuos/input_airbrush.asp). It is an optionally available stylus for Wacom's Intuos line of pressure sensitive tablets for pen based input and features a finger wheel, similar to a scroll-wheel, with 1024 levels of activation and a side switch. The purpose of the finger wheel on this stylus is to provide artists with the possibility to control ink-flow in graphical applications, as is the case with real airbrushes.

Because of the fact that menu and tool selection is essentially a 1 DOF operation and many previous menu techniques suffered from difficult-to-learn selection methods due to their 3 DOF nature, a scroll-wheel should be well suited to this task, as it is naturally constrained to 1 DOF and provides an input stream utilizable for navigation in a one-dimensional space. Moreover, the input is in fact discrete and single "notches" in the wheel provide passive-haptic feedback when switching from one position to the other, making selection from a set particularly easy. By studying the use of scroll-wheels on mice, we learned that these sets can be the parameter space of an (pseudo-)continuous value (e.g. document position, ink-flow, zoom-level) or a discrete set of options and tools. We want to use the scroll-wheel on the stylus to switch between the following elements according to our terminology of system control tasks:

- *Tools* and *modes* can be changed which results in tool and mode switches when the scroll-wheel is operated.
- Single *commands* and *parameters* can be selected that can be executed or modified thereupon. If a set only consists of commands and parameters, navigation between these corresponds to navigation in a conventional one-dimensional menu.
- The space for navigation with the scroll-wheel can also be the *parameter space* of a scalar parameter itself.

A single set of tools, commands and parameters to choose from would only be acceptable for a limited number of elements, thus the implementation of the user interface should offer some sort of hierarchy and organization of tools and menu items using the input of two buttons on the pen to move between levels in the hierarchy.

4. IMPLEMENTATION

4.1. User Interface

We use two different styles of visual feedback: Toolbars and cylindrical menus.

In toolbars, 3D icons and labels are used to represent tools and commands. These icons can also reflect the parameterization of the tool, e.g. line size. Captions of commands or parameters are represented by billboarded labels. Apart from that, labels are also used to clarify the meaning of icons and appear above them in case they are needed by showing the name of the tool. It is common to both styles that the item currently selected – tool or menu entry – is always visible in front of the tip of the tracked pen (see Figure 2 and Figure 3 for representation styles).

In a toolbar, the 3D icons and labels are positioned on an imaginary linear list aligned parallel with and running through the entire length of the pen (Figure 2). Since the scroll-wheel on the pen is also aligned this way, turning of the wheel naturally maps to pushing away and pulling near elements in this linear list. Since the list of icons and labels would interfere with the scene, partly be obstructed by the pen, and thus distract the user if entirely visible, only the entry

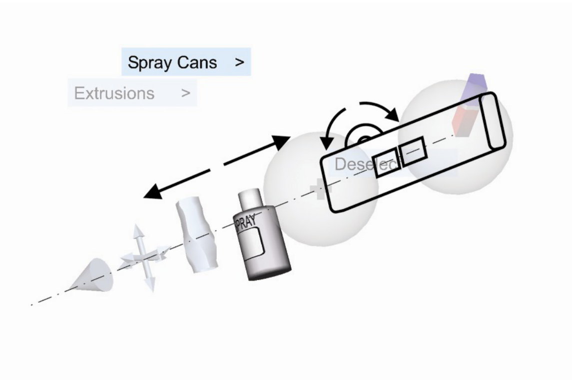


Figure 2: Toolbar representation of the user interface. Only the icon of the selected tool is actually displayed – semi-transparent icons in the illustration are added for the sake of clarity.

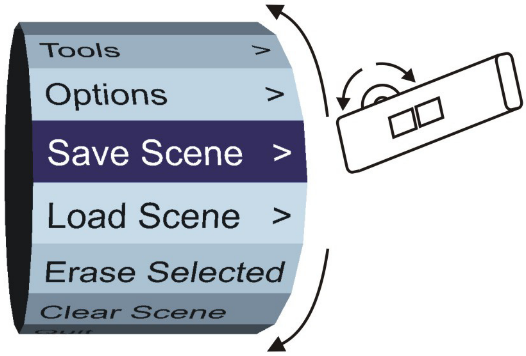


Figure 3: The graphical cylindrical menu representation of the user interface.

currently selected is shown in front of the tip of the pen. In order to maintain the sense of spatiality and to give the impression that the icons and labels are actually beaded on a line, icons and labels are sliding in and out the tip of the pen – fading in and out while they do so – in the direction of the rotation of the scroll-wheel. According to [Bed99a] this animated transition should also help users to better remember positions of tools in the toolbar. If the scroll-wheel is rotated by several notches in a short time and many entries in the list have to be skipped, the speed of the animation of the icons and labels sliding in or out at the tip of the pen is accelerated to allow for faster navigation [Hin02a].

A cylindrical menu is used for sets of application control tasks that are made up entirely of commands and parameters. In 2D desktop environments these are usually implemented as pull-down menus. As the turning of a scroll-wheel naturally maps to rolling of a horizontal cylinder, the use of a cylindrical rotational menu instead of a list on a plane is strongly suggested (Figure 3). The advantage of this representation over our toolbar style representation obviously is the visibility of several entries at once. This eases the process of identifying and targeting the desired item and significantly speeds up navigation. As the user turns the scroll-wheel the menu will rotate to bring the next or previous entry to the tip of the pen. As is the case with the animation of the toolbar fast scrolling of the wheel results in faster rotation of the menu. In addition to the rotation of the menu, the selected item will also get highlighted to be recognized at a glance. As the menu is shaped like a cylinder it always has the same size, regardless of the number of entries, which avoids cluttering of the workspace. As with labels, the cylindrical menu is view dependent and will always face the user. This ensures its readability, no matter how the pen is oriented.

As mentioned above, due to the large number of tools, application commands and parameters in even moderate complex applications, some sort of grouping and hierarchy is necessary. Two buttons on the pen – one being the main button every stylus features, the other being a supplementary button – are used to navigate between toolbars and menus and submenus of arbitrary depth, respectively. In order to conserve the reference of a submenu, its parent stays visible in its current state at one side of the submenu, much like in conventional 2D pull-down menus (Figure 4).

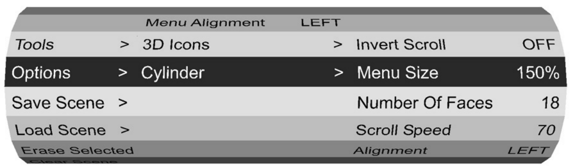


Figure 4: Example of a graphical cylindrical menu showing the root menu with two cascaded submenus. Menu items containing a submenu are indicated by the '>' character.

4.2. Control

The two buttons of the Wheelie are not only used for navigating the menu and tool hierarchy, but also for activation of items and direct manipulation:

- *Primary (activation) button*: If the item currently selected in the system control hierarchy is a single tool, the primary button is used to apply the tool while being pressed. If a command is selected, it is executed when the primary button is activated. If the selected item is a parameter, it can be adjusted by scrolling through the parameter-space with the scroll-wheel while the primary button is pressed. Should the entry currently selected contain a subset of tools or a submenu, activation of the primary button effectuates a

 ISBN 1213-6972 ISBN 80-86943-09-7

change to the subset and descends in the menu hierarchy.

- *Secondary (escape) button*: The supplementary button on the pen is used to ascend one level in the hierarchy structure.
- *Scroll-wheel*: The scroll-wheel is mainly used to browse through the current set of available tools, commands and parameters, or to select an item from the graphical menu. As mentioned above, in combination with the primary button it allows parameterization of the current tool or menu item. For instance, the diameter of the cross-section of an extrusion tool, transparency or the "ink-flow" of a spraying tool can be changed during direct manipulation activities.

4.3. Hardware Setup

In order to build a prototype Wheelie stylus we used a commercially available marker and the electronic parts and the scroll-wheel of an old mouse. As it turned out, the housing of a marker is very well suited for our purpose, since it provides enough space for the integration of a scroll-wheel in the narrow part of the grip as well as two buttons on the flat side of it. The scroll-wheel is installed in a position that, when the pen is held as if for writing, allows comfortable operation with the tip of the index finger. In the place where the thumb is resting against the flat part of the pen, two buttons are mounted in a row insofar as the thumb does not have to be moved to reach either one of them. This configuration resembles the handling of a mouse with two extra buttons on its side, used for document history navigation in internet browsers or the like. Since it is crucial for the user interface to deliver a smooth navigation experience, it is important that both buttons can be operated easily. Since push buttons are often awkward to activate if not pressed directly from above, we decided to use buttons that tilt forward (front button) and backwards (back button). This significantly enhances button control with the thumb. The button closer to the tip of the pen functions as the primary activation button whereas the rear button acts as the secondary escape button. Unlike the scroll-wheels on a mouse, the wheel on our prototype pen does not feature the additional function as a button. Although this would yield an extra degree of freedom, the force needed to activate it with the index finger is uncomfortable since the pen is held in hand and does not reside on a flat surface that provides resistance to this force in opposite direction. This is especially true since the force to activate a button click on the scroll-wheel must be higher than on ordinary buttons to avoid involuntary activation while operating the scroll-wheel.

The test environment consisted of a Barco Baron™ virtual table, an ART optical tracker system and a PC with a GeForce graphics card.

5. EVALUATION

To see how Wheelie performs in practice and to informally observe users while interacting in a complex application with it, we developed a test application allowing the user to select from a variety of tools, commands, and parameters. The test application, implemented in the Studierstube framework [Sch02a], resembles a desktop graphic application in which the user can spray objects, create extrusions from cross-sections and draw lines. Additionally, common editing tasks such as selecting, moving, scaling, painting and deleting of object parts are available as tools. The graphical menu offers commands to erase, save and load created scenes, as well as options and parameters to customize the user interface.

The concept of scrolling through a list of available tools with the pen was well received and the testers did not voice problems with the overall handling and

Figure 5: Spraying in the scene with a spray can shaped tool. The icon also reflects the aperture angle of the tool.

Figure 6: With the move tool objects not only can be moved but also scaled by using the scroll-wheel while the primary key is pressed.

navigation of the interface (Figure 5 and Figure 6).

The use of the scroll-wheel during the application of the tool, e.g. the change of the aperture angle of a spray can while spraying, was less obvious, but immediately adopted by the users when demonstrated. This, however, revealed another issue: Users often desired to adjust the parameter before actually applying the tool. While this does not pose a problem if the tool is only interacting with objects in the scene, it was a nuisance when the tool was actually adding new content. Besides, it is disputable which parameter of a tool should be subject to change in the course of a workflow. Another issue worth mentioning emerged from using our interface in combination with a virtual palette: Widgets on the palette must be operated with a simple pointing tool, having no function of its own. Since this would make recurrent switches between the pointer and actual tools necessary, it is solved by simply deactivating the function of a tool when interacting with a widget.

An interesting aspect was noted when we observed control of the graphical cylindrical menu system with the Wheelie: Although the cylindrical shape of the menu strongly suggests that its affordance is to be rotated using the scroll-wheel in the same direction, many users thought of it to work as moving the highlighted selected item with the scroll-wheel as opposed to turning the menu cylinder so that the desired entry gets aligned with the tip of the pen. We think this is mainly induced by the visual feedback that highlights the selected item and the notion of graphical menus to be static, gathered from experience during extensive use of 2D desktop menu systems. Tests with less experienced users should be made to investigate this theory. Nevertheless, we added the option of inverting the scroll-direction in the menu representation of the interface (Figure 7).

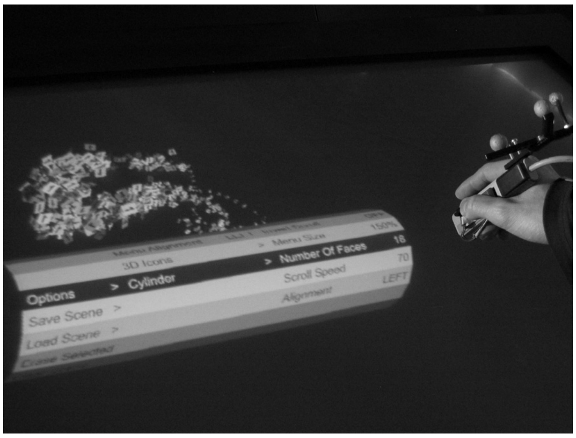


Figure 7: The graphical menu in action. Rotation direction of the menu when scrolling through entries can be set to user's preference.

Another point mentioned by our testers was that the smooth animation (i.e. rotation) of the cylindrical menu is dispensable. While the animation helps in the toolbar representation as only one item is visible at a time, the cylindrical menu provides an overview of surrounding entries and transitions need not be animated. On this account we implemented an "expert mode" (in both representations) that performs instantaneous jumps when operating the scroll-wheel. The default configuration was changed to smooth transition in the toolbar representation and instantaneous transition in the graphical cylindrical menu.

On the hardware side, a wireless device is preferable to our wired prototype of the Wheelie. Therefore future versions should only be made on a wireless (e.g. Bluetooth) basis to further ease the handling of the Wheelie.

6. CONCLUSION AND FUTURE WORK

Wheelie, a scroll-wheel enhanced tracked stylus for VEs presented in this paper, is an input-device that provides application and user interface designers with a separate one-dimensional discrete input stream applicable to various system control techniques for complex VE applications. The user interface and menu system proposed in this work allow the user to quickly change between different tools, execute commands and perform parameter manipulation with the tips of his fingers of just a single hand. It takes advantage of the fact that the input of the scroll-wheel is constrained to 1 DOF, which makes it perfectly tailored to the task of menu selection and tool switching, and certainly better than continuous input streams like a joystick for these purposes.

When compared to other 3D system control techniques, the Wheelie exhibits true general purpose properties:

Although more extensive experimental evaluation will have to be performed, switching between single different tools for direct manipulation is not as fast as ultimately possible with the ToolFinger approach which, on the other hand, is a special purpose technique limited to a single set of available tools. A perfect combination would be the use of entire ToolFingers as entries on the toolbar representation to choose from with the scroll-wheel.

The C^3, designed as a quick-access menu, provides a faster way of activating commands but is also limited to a single set. The Wheelie menu system offers a familiar 2D-like representation of a hierarchical graphical menu, providing a virtually limitless space for commands while at the same time sustaining its size, not occluding much of the valuable visible

 ISBN 1213-6972 ISBN 80-86943-09-7

space. Other interaction techniques can still be used alongside Wheelie since Wheelie only extends the input capabilities of a stylus.

Quantitative analyses which compare Wheelie to other types of similar system control techniques will have to be made in order to get proper information on how Wheelie performs against them. Nevertheless, we can confidently assume that a scroll-wheel implements a useful addition to the conventional 3D stylus without cluttering the interface or complicating traditional stylus interaction.

7. REFERENCES

[Bed99a] Bederson B. B., Boltman A.: Does Animation Help Users Build Mental Maps of Spatial Information? In INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization (Washington, DC, USA, October 1999), IEEE Computer Society, pp. 28-35.

[Bow99a] Bowman D. A., Hodges L.: Formalizing the Design, Evaluation, and Application of Interaction Techniques for Immersive Virtual Environments. In The Journal of Visual Languages and Computing 10, 1 (1999), pp. 37-53.

[Bow01a] Bowman D. A., Wingrave C.A.: Design and Evaluation of Menu Systems for Immersive Virtual Environments. In VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01) (Washington, DC, USA, 2001), IEEE Computer Society, p.149.

[Coq99a] Coquillart S., Wesche G.: The Virtual Palette and the Virtual Remote Control Panel: A Device and an Interaction Paradigm for the Responsive Workbench(TM). In VR '99: Proceedings of the IEEE Virtual Reality (Washingron , DC, USA, 1999), IEEE Computer Society, p. 213.

[Cut97a] Cutler L. D., Fröhlich B., Hanrahan P.: Two-handed Direct Manipulation on the Responsive Workbench. In SI3D '97: Proceedings of the 1997 symposium on Interactive 3D Graphics (New York, NY, USA, 1997), ACM Press, p.107 ff.

[Gro01a] Grosjean J., Coquillart S.: Command & Control Cube: A Shortcut Paradigm for Virtual Environments. In Immersive Projection Technology and Virtual Environments 2001 Proceedings (May 2001), pp. 1-12.

[Hin02a] Hinckley K., Cutrell E., Bathiche S., Muss T.: Quantitative Analysis of Scrolling Techniques. In CHI '02: Proceedings of the SIGCHI conference on Human Factors in Computing Systems (New York, NY, USA, 2002), ACM Press, pp. 65-72.

[Kru00a] Kruifjj E.: System Control. In 3D User Interface Design: Fundamental Techniques, Therory, and Practice. SIGGRAPH 2000 Course Notes (July 2000), Bowman D. A., Kruijff E., LaViola Jr. J., Mine M., Poupyrev I. (eds.), pp. 147-165.

[Lia94a] Liang J., Green M.: JDCAD: A Highly Interactive 3D Modeling System. Computers and Graphics 18, 4 (July 1994), pp. 499-506.

[Lin99a] Lindeman R. W., Sibert J. L., Hahn J.K.: Hand-held Windows: Towards effective 2D Interaction in Immersive Environments. In VR (1999), pp. 205-212.

[McM97a] McMillan G. R., Eggleson R. G., Anderson T. R.: Nonconventional controls. In Handbook of Human Factors and Ergonomics 2nd ed., Salvendy G. (ed.) (New York, NY, USA, 1997), John Wiley & Sons, pp. 729-771.

[Min97a] Mine M. R., Brooks Jr. F. P., Sequin C. H.: Moving Objects in Space: Exploiting Proprioception in Virtual-Environment Interaction. In SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (1997), ACM Press/Addison-Wesley Publishing Co., pp. 19-26.

[Min97b] Mine M. R.: Isaac: A Meta-CAD System for Virtual Environments. Computer-Aided Design 29, 8 (1997), pp. 547-554.

[Sch99a] Schmalstieg D., Encarnação L. M., Szalavári Z.: Using Transparent Props for Interaction with the Virtual Table. In SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics (New York, NY, USA, 1999), ACM Press, pp.147-153

[Sch02a] Schmalstieg D., Fuhrmann A.L., Hesina G., Szalavári Z., Encarnação L. M., Gervautz M., Purgathofer W.: The Studierstube Augmented Reality Project. PRESENCE: Teleoperators and Virtual Environments 11, 1 (February 2002).

[Ste03a] Stefani O., Hoffman H., Rauschenbach J.: Design of Interaction Devices for Optical Tracking in Immersive Environments. HCI International 2003 Conference Proceedings (2003), Lawrence Erlbaum Associates, Inc.

[Sza97a] Szalavári Z., Gervautz M.: The Personal Interaction Panel – A Two-handed Interface for Augmented Reality. In Proceedings of EUROGRAPHICS '97 (September 1997), pp. 335-346.

[Wes03a] Wesche G.: The Toolfinger: Supporting Complex Direct Manipulation in Virtual Environments. In EGVE '03: Proceedings of the Workshop on Virtual Environments 2003 (New York, NY, USA, 2003), ACM Press, pp. 39-45.

GPU-Friendly High-Quality Terrain Rendering

Jens Schneider
Computer Graphics and Visualization Group
Technische Universität München
Lehrstuhl I15, Boltzmannstraße 3
D-85748 Garching bei München
jens.schneider@in.tum.de

Rüdiger Westermann
Computer Graphics and Visualization Group
Technische Universität München
Lehrstuhl I15, Boltzmannstraße 3
D-85748 Garching bei München
westermann@in.tum.de

ABSTRACT

We present a LOD rendering technique for large, textured terrain, which is well-suited for recent GPUs. In a pre-process, we tile the domain, and we compute for each tile a discrete set of LODs using a nested mesh hierarchy. This hierarchy can be encoded progressively. At run time, continuous LODs can simply be generated by interpolation of per-vertex height values on the GPU. Any mesh re-triangulation at run-time is avoided. Because the number of triangles in the mesh hierarchy is substantially decimated and by progressive transmission of vertices, our approach significantly reduces bandwidth requirements. During a typical fly-over we can guarantee extremely small pixel errors at very high frame rates.

Keywords

Terrain rendering, hierarchical meshing, GPUs, progressive data transfer, geomorphs

1. INTRODUCTION

Despite the advances in CPU and GPU technology, for the largest available terrain data sets rendering techniques still cannot run at acceptable rates *and* quality. As processing, memory, and bandwidth capabilities continue to increase, so does the resolution of scanned landscapes and recent display technology. Today, satellite range scans comprised of over a billion of samples are available, making even the handling of such data sets difficult to perform due to memory constraints. In addition, high resolution displays of about 10 Mpixels [IBM] demand a substantial increase in the number of primitives to be transferred to and processed on the GPU. The requirements imposed by current and future data acquisition and display technology make real-time visualizations difficult to perform on even the most powerful workstations. Therefore, the need for a terrain rendering system that comprehensively addresses the aforementioned issues is clear.

Journal of WSCG ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

2. RELATED WORK

From a high-level view, previous approaches for terrain rendering can be classified into the three following categories.

View-dependent refinement

View-dependent refinement methods construct a continuous LOD triangulation on the CPU with respect to a given world- and screen-space error. Gross et al. [GGS95] employ a wavelet decomposition to generate adaptive quadtree meshes for terrain data, combined with a lookup-table to store an irregular triangulation for each of the possible quadtree leafs. Pajarola [Paj98] introduced *restricted* quadtrees [HB87] for terrain rendering. Duchaineau et al. [DWS+97] used triangle bintrees to perform the remeshing. Triangulated irregular networks (TINs) where first proposed by Peucker et al. [PFL78], and later automated by Fowler et al. [FL79]. Garland et al. [GH95] employed a greedy insertion strategy to construct a TIN. Progressive meshes (PMs) were modified with respect to the demands in terrain rendering by Hoppe [Hop98].

To speed up the remeshing process, frame-to-frame coherence can be exploited. Priority queues that can be updated incrementally to guide the remeshing are one alternative [DWS+97]. A different approach updates a quadtree data structure incrementally to keep track of vertex dependencies [LKR+96]. Hoppe proposed a method that keeps active cuts to achieve an incremental update [Hop98]. While the exploitation of frame-to-frame coherence usually results in a reason-

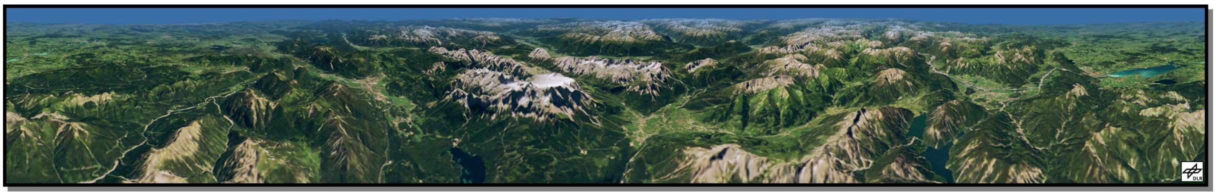


Figure 1: A 360° panorama of the Alps (7K×1K pixels), generated with our method in less than 4 seconds. This time includes rendering, reading data back from the GPU, and writing the final image to the disk.

able speed up, for particular camera movements such as shoulder views in an airplane simulation a considerable loss in performance can be observed. Furthermore, frame-to-frame coherent approaches are usually harder to implement due to LOD constraints. This was recognized by Lindstrom et al. [LP01, LP02], who proposed a simple to implement, yet efficient method to rebuild the mesh from scratch in every frame. They improve the error metric proposed by Blow [Blo00].

If the terrain gets excessively large, many of the mentioned algorithms choose to partition the terrain into square blocks or chunks of data, which can then be processed independently from each other [KLR+95, SN95]. The advantage is that these chunks can also be paged independently. However, care has to be taken to avoid invalid vertices (so-called T-vertices) at chunk boundaries. One elegant approach to avoid these invalid vertices in a quadtree was taken by Röttger et al. [RHSS98]. By restricting the error metric, they automatically guaranteed a valid mesh. However, a generalization to chunked meshes is not trivial and would also limit the error metric to a Manhattan distance.

More recently, Ulrich [Ulr00] suggested to use restricted quadtree meshes without boundary constraints for the chunks, and to fill possible cracks between them using *flanges* or *skirts* – fins of geometry along the boundaries pointing downwards from the terrain. However, ensuring correct anisotropic texture filtering at these boundaries is not trivial due to the different viewing angle. A more general approach is to stitch boundaries together using so-called zero-area triangles (also called *ribbons* in [Ulr00]), which guarantees correct filtering.

Pomeranz [Pom00] suggested to use clusters of ROAM triangulations (RUSTiC). To ensure validity, clusters are enforced to uphold an edge constraint: on shared edges the clusters must share vertices exactly. This approach is also one of the first terrain rendering algorithms exploiting graphics hardware. RUSTiC achieves improved performance over ROAM by rendering clusters as triangle strips. Hwa et al. [HDJ04] used 4-8 meshes that induce a diamond-based hierarchy on both textures and the height field. Combined with a space-filling curve memory layout this allows for efficient out-of-core rendering of the terrain, utilizing GPU memory as a cache. However, since each other texture level is rotated by 45°, a costly update of vertex texture coordinates has to be performed.

Pre-computed geometry batches

Based on the observation that on recent GPUs the time that is saved by rendering less triangles due to adaptive re-triangulation is entirely amortized by the time needed to perform the re-triangulation, several authors suggested to pre-triangulate the input data as much as possible. Cignoni et al. [CGG+03a] suggested to replace triangles in the remeshing process by a *batch*, a new primitive that approximates the terrain over a triangular part of the input domain using a pre-computed TIN. Stripping these TINs prior to rendering made them highly efficient. Batches were kept in a bintree, for which usual run-time re-meshing is performed, hence the name of the method: Batched Dynamic Adaptive Meshes (BDAM).

In [CGG+03b], the authors improved on their previous work to successfully render planet-size meshes at interactive rates. Their system does not support geomorphs, but a screen-space error of one pixel for a 640×480 view port can usually be guaranteed. However, this could become a problem soon, as displays are about to reach 10Mpixels. Consequently, considerably more triangles would have to be rendered to meet a given screen-space error.

Non-adaptive triangulation

Only very recently, Losasso et al. [LH04] took full advantage of the speed of current consumer class GPUs. They abandoned any view-dependent remeshing in favor of so-called geometry clipmaps, a triangulation that is approximately screen-space uniform. Specifically they used concentric, uniformly tessellated, square patches around the camera dropping exponentially in resolution with distance. During run time, geometry is fetched from a toroidal buffer residing on the GPU. The update of this buffer is done by the CPU.

Since the heighfield raster data is used directly, it can be compressed very efficiently. By applying a compression scheme derived from Microsoft's WMV format [Mal00], compression ratios of up to 100:1 can be achieved. Because decoding the compressed data puts a considerable amount of work on the CPU, the decoder can eventually fall behind faster camera motions, resulting in a blurry representation of the terrain. Despite the fact that geomorphs are not an issue for this system, both the screen-space and world-space errors are hard to control, implying an rms of about 1.5m. Optimal geometry filtering cannot be performed due to the screen-space aligned topology. Also, since height fields compress a lot better than regular images, the application of photo textures will most likely result in a major increase in memory requirements. Still, extremely high frame rates for virtually arbitrarily large data sets can be achieved using this method.

3. CONTRIBUTIONS

In this work, we combine the advantages of continuous LOD semi-regular meshes with the advantages of a discrete LOD hierarchy, thus avoiding any re-triangulation at run-time. In contrast to BDAM we also avoid expensive irregular triangulations, greatly improving pre-processing from several hours to some minutes. The proposed method generates high quality renderings by supporting a continuous LOD representation including geomorphs and photo-texturing. In contrast to previous methods, the terrain is guaranteed to be refined within a user-defined screen- *and* world-space error. Aliasing is avoided by employing optimal geometry filtering, at the best possible geometric resolution. At run-time, discrete sets of decimated mesh structures are transmitted progressively, resulting in high bandwidth efficiency. To obtain a continuous LOD representations, these sets are interpolated and rendered using functionality on recent GPUs.

Algorithm overview

The domain is first partitioned into a set of equally sized tiles. For each tile, a discrete set of LODs is computed by means of a nested mesh hierarchy. The construction of such a hierarchy is described in section 4. This hierarchy has several beneficial properties: Firstly, for each level of the mesh the terrain is decimated according to a given world-space error, reducing the total amount of triangles. Secondly, to compute a continuous LOD representation, vertices at finer resolutions only have to be morphed in height onto the next coarser level. Third, as the hierarchy is nested, each finer level is represented by all vertices at coarser levels plus additional vertices required to resolve the current level properly. These additional vertices can be transmitted progressively.

The terrain hierarchy, including per-vertex morph values, is then prepared for rendering on the GPU. The particular data structure used is discussed in-depth in section 5. For textures, the S3TC standard is employed, which enables high-resolution mipmaps to be used. All data is stored in vertex buffers and 2D textures that are handled by a memory manager to minimize bus transfer. This issue is subject of section 6.

4. NESTED MESH HIERARCHY

The most common way to avoid sampling artifacts in terrain rendering is by means of a LOD representation. Such a hierarchy can either be represented implicitly by adaptive re-triangulation at run time, or it can be explicitly pre-computed for discrete LOD levels.

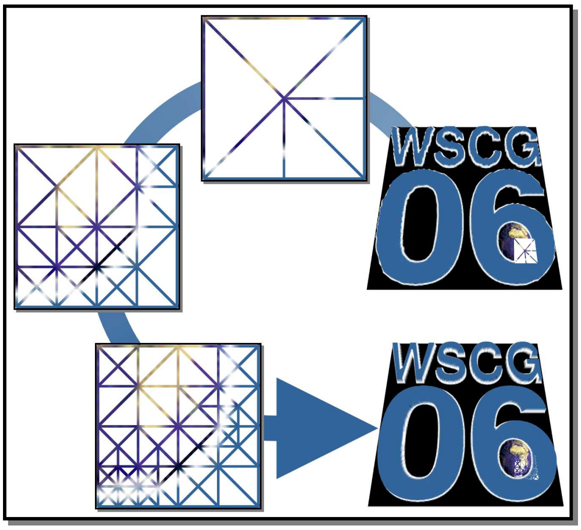


Figure 2: Levels of the nested mesh hierarchy.

A given height field $H : \mathbb{N}^2 \mapsto \mathbb{Z}$ can be approximated by a triangular mesh parameterized over a 2D domain. The surface of this mesh defines a reconstruction H' of H. The approximation quality of the mesh is then measured by a point-wise error metric $\delta : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, extended to the entire domain. In the current work, we use the canonical extension of the $L_{\max}$ error metric to measure the error between H and H':

$$\delta(H, H') := \max_{x,y} \ \delta(H(x,y), H'(x,y))$$

By generating approximations of the height field with decreasingly lower approximation error, a mesh hierarchy that represents the original terrain at ever finer scales is constructed. The hierarchy employed in this work is *nested* with respect to the triangulation: For each triangle on level i with canonic parameterization Ω_i there is a triangle on the next coarser level $i-1$ with parameterization Ω_{i-1} such that $\Omega_i \subseteq \Omega_{i-1}$. That is, if both triangles are projected onto the domain, the triangle at level i is contained entirely in the triangle at level $i+1$. Such a hierarchy is automatically generated by restricted quadtree [HB87, Paj98], bintree [DWS$^+$97] or red-green refinement [BSW83].

To generate a discrete set of nested hierarchy levels, the terrain is partitioned into equal tiles of size 257^2, with an overlap of one sample in either direction. Then, an error vector $(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{n-1})$ of exponentially decreasing entries $\varepsilon_i := 2^{n-1-i}$ is built, where the ε_i are

 ISBN 1213-6972 ISBN 80-86943-09-7

usually measured in meters or feet. The particular choice is motivated in section 5. Starting with ε_0, a hierarchy $\{\mathcal{M}_i\}_{i=0}^{n-1}$ of restricted quadtree meshes satisfying $V_i \subseteq V_{i+1}$ and $\varepsilon_{i+1} \leq \delta(H_i', H) \leq \varepsilon_i$ is constructed. Here V_i and V_{i+1} are the sets of vertices at hierarchy levels i and $i+1$. More precisely, in a top-down approach we construct each $\mathcal{M}_{i+1}$ by refining $\mathcal{M}_i$, and we stop the construction if $\delta(H_{i+1}', H) \leq \varepsilon_{i+1}$.

To generate the next finer hierarchy level from the current mesh, recursive quadtree refinement is performed until one of the following two conditions is met.

1. the maximum deviation between the new mesh and the original terrain is less than the error threshold defined for the level.
2. the spacing between vertices of the mesh becomes smaller than the error threshold defined for the level.

The second criterion is enforced by prohibiting the quadtree from being refined below a certain scale. This weakens the requirement $\varepsilon_{i+1} \leq \delta(H_i', H) \leq \varepsilon_i$, but generally $\delta(H_i', H)$ is still less than ε_i. In this way we can avoid aliasing artifacts due to subsampling along the domain axes. In a second step (following the Push/Pull paradigm), geometry changes are propagated from fine to coarse and sub-quadtrees are refined where needed to avoid T-vertices.

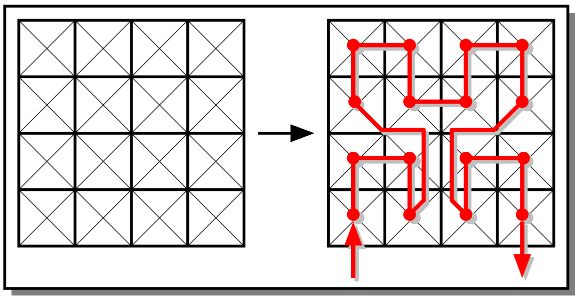


Figure 3: Quadtree mesh and Π-order traversal.

The quadtree is then decomposed into recursive triangle fans [RHSS98] or a single triangle strip [LP02]. Using triangle strips is possible in our framework, but generating them increases the time spend for pre-processing considerably. Triangle fans, on the other hand, are easy to implement, reduce meshing time and are similarly cache friendly as strips. However, generating fans results in a lot of separate primitives. In order to render these primitives efficiently, primitive restarts are employed. Primitive restarts are available on recent nVidia GPUs and are exposed in OpenGL by the GL_primitive_restart_NV extension. When rendering indexed vertices, the user may define a special index. Whenever this index is encountered, no vertex is fetched but instead a new primitive is started. This allows for a list of fans to be rendered efficiently by using only a single draw call, reducing state changes and setup overhead. To generate fans the quadtree is traversed recursively in depth-first order. As a result, we visit each fan in the order of a Π-order space-filling curve (see figure 3), which was successfully used in [LP02] to linearize memory layouts. This traversal has the nice property that fans generated after each other have a very high probability to be adjacent (in a full quadtree all consecutive fans are adjacent), in which case the newer one can re-use two or even three vertices of the previous one. Since each fan has at most 9 vertices, the last fan will always be cached entirely on current GPUs.

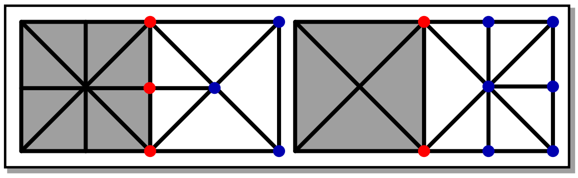


Figure 4: Best and worst cases for vertex cache re-usal of fans. The gray fan can re-use the red vertices of the white fan, resulting in a cache coherence of at least 25%

Thus, recursive fans can re-use between $2/8$ and $3/6$ of their vertices (see figure 4).

To obtain a continuous LOD representation, we interpolate between the discrete LODs $\mathcal{M}_i$. This is known as *Geomorphing* [FEKR90]. In a nested hierarchy, vertices retain their position within the domain during morphing. Due to the property $V_i \subseteq V_{i+1}$ each vertex at level i thus stores one height value for level i and each coarser level $k < i$. To render a LOD between two consecutive levels, the triangle mesh at the finer level is rendered and vertices are morphed accordingly. Although higher order interpolation is possible, only linear interpolation is considered in this work for efficiency reasons. This is described later in more detail.

5. RENDERING FRAMEWORK

As a benefit of the nested mesh hierarchy, tiles can be uploaded progressively to the GPU. On the GPU, an appropriate data structure accommodates real-time rendering at high quality, including photo-texturing. Optionally, if high resolution view ports require the screen space error to be increased, geomorphing is performed on the GPU. At the same time, the CPU performs view frustum culling and level of detail computations on a per-tile basis. Since all GPU tasks are programmed in a high-level shading language, the entire framework is extendable and can easily be tailored to fit custom needs.

GPU data structures

As soon as a particular tile has to be rendered, a vertex buffer large enough to store all shared vertices of that

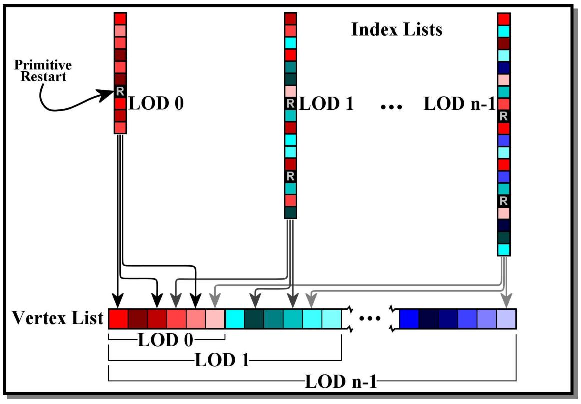


Figure 5: The GPU data structures used to enable progressive transmission of vertices and indices.

tile is created. In this buffer, vertices are organized in blocks according to their respective hierarchy levels. (see figure 5). The associated topology is stored in one separate element array for each level. The i^{th} element array contains only indices into the first $i+1$ blocks of the vertices. Such a shared vertex representation has two major advantages. Firstly, it reduces storage requirements compared to non-shared representations. This is of special importance when additional vertex attributes, such as geomorphs have to be stored. Secondly, it enables progressive transmission by re-using vertices of coarser levels.

Because the tiles used in this work always have a resolution of 257^2, relative domain coordinates are encoded in 9 bits. The height value can be considerably larger. It is therefore encoded using 14 bits. All three values are stored in two 16 bit vertex attribute components. They are decoded in the vertex shader during rendering.

If geomorphs are enabled, additional storage requirements arise. The method is still memory efficient, as only one additional height value per coarser level needs to be stored. Since usually only small offsets to the original height are needed, 8 bits per value are sufficient. This allows us to morph vertices within a range of $+127\ldots-128$ units.

Run time processing

For each tile we keep an axis-aligned bounding box to accommodate view frustum culling on the CPU. For every frame, visible tiles are depth-sorted to exploit the early-depth test, if available, and to reduce overdraw. A memory manager, which is described below, ensures that all visible tiles can be rendered by paging in data not yet resident on the GPU.

Then for each visible tile the appropriate LOD is computed. The index buffer as well as the vertices required to render the respective level are sent to the GPU, if not already resident. If a tile has been rendered previously, at least a subset of vertices has already been sent to the GPU. In this case, only the remaining vertices required to render the current level are transmitted and written to the respective vertex buffer on the GPU. In this way, even though an array large enough to keep all vertices has to be allocated on the GPU, bandwidth requirements at run time are substantially reduced.

To avoid cracks at tile boundaries, neighboring tiles are stitched together using zero-area triangles. For each tile and each level in the hierarchy, the set of border vertices along with all attributes is duplicated in system memory. Whenever two neighboring tiles are visible, the necessary triangles to fill out T-junctions are generated on the CPU and are then rendered. Since this process uses exact duplicates of the vertices on the GPU and the same GPU programs are employed, cracks are avoided without numerical precision issues.

Level of detail

Determining the appropriate LOD for each tile and vertex requires the projection of the user-defined pixel error to object space. Previous approaches rely on conservative estimates of this error and are often equivalent to a linear approximation of the projection. Since such estimates usually over-estimate the error, even for pixel errors larger than one aliasing might still occur. We compute a more precise error metric by directly using the current projection matrix, which maps homogeneous object coordinates $v=(v_1,v_2,v_3,1)$ to screen-space coordinates $s=(s_1,s_2,s_3)$. Here, s_3 corresponds to the depth value. The appropriate scale of details ρ can then be computed in a similar way as the appropriate mipmap scale for texturing [Wil83]:

$$\rho := \sqrt{\frac{\sum_{i=1}^{3}\left(\frac{\partial v_i}{\partial s_1}ds_1+\frac{\partial v_i}{\partial s_2}ds_2\right)^2}{ds_1^2+ds_2^2}}$$

To compute ρ, s is expressed in parametric form $s(v)$, already including perspective division and scaling of the canonic frustum to pixel coordinates. The Jacobi matrix at v consists of the partial derivatives $J_{ij}(v) := \partial s_i/\partial v_j$. The inverse transpose of $J(v)$ contains exactly the partial derivatives required to compute ρ. The differentials ds_i are required to map from units of the height field (*eg.*,feet or meters) to pixels. Computing ρ yields the optimum scale corresponding to a screen-space error τ equal to 1 pixel. If the user selects a different screen-space error, the frustum is scaled to pixel coordinates divided by τ instead of using the entire resolution. Then, ρ is the object space distance that projects onto τ pixels.

On the CPU, ρ_j is computed per tile for each corner j of its bounding box. Because entries of the error vector are given by $\varepsilon_i = 2^{n-1-i}$ units, the optimum LOD value is computed by $\lambda_j := \lambda_{max} - \lfloor \log_2(\rho_j) \rfloor$, where

 ISBN 1213-6972 ISBN 80-86943-09-7

$\lambda_{max} = n-1$ is the number of available levels. The mesh $\mathcal{M}_{\min_j\{\lambda_j\}}$ is then selected for rendering the tile.

Geomorphing

As mentioned before, high resolution displays coupled with a low screen-space error can require most of the terrain to be rendered at the highest resolution. In order to maintain stable and interactive frame rates, the tolerable screen-space error has to be increased. To prevent popping artifacts, geomorphs are applied. For every vertex v in a tile, the λ_j at box corners are trilinearly interpolated on the GPU to get an approximate vertex LOD $\lambda(v)$. Geomorphing [FEKR90] now consists of linearly interpolating height values $H_{\lfloor\lambda(v)\rfloor}$ and $H_{\lfloor\lambda(v)\rfloor+1}$,using the fractional part $\lambda(v) - \lfloor\lambda(v)\rfloor$ as interpolation weight.

Finding the correct height values on the GPU could be implemented in a straight forward manner using conditionals. As conditionals are costly on current GPUs, we avoid them by implementing a different approach based on clamped forward differences. In this approach, we treat height values $\{H_i\}_{i=0}^{n-1}$ as the control points of a piecewise linear interpolant in λ. To

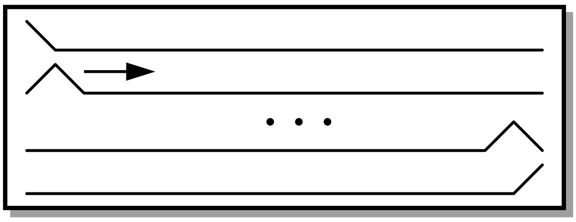


Figure 6: Basis-functions η' for geomorphs.

obtain $H(\lambda)$, we compute shifted basis-functions that can be reduced using simple dot product arithmetic. Firstly, we compute a vector-valued function

$$\eta(\lambda) := clamp\left((\lambda,\lambda,\lambda,\lambda,\ldots)^t - (0,1,2,3,\ldots)^t, 0, 1\right)$$

Each component i of η contains a linear ramp between $\lambda = i$ and $\lambda = i+1$. For $\lambda \le i$ it is 0, and for $\lambda \ge i+1$ it is 1. Then, the desired basis function is obtained by computing forward differences on η:

$$\eta_i'(\lambda) := \begin{cases} 1-\eta_0(\lambda) & \text{if i=0} \\ \eta_{i-1}(\lambda) - \eta_i(\lambda) & \text{else} \end{cases}$$

Finally, the η_i' contain the well-known basis functions for linear interpolation (see figure 6). Interpolation can now be written as the dot product $H(\lambda) = \sum_{i=0}^{n-1} \eta_i'(\lambda) \cdot H_i$. This method is highly efficient on the GPU and in our case ($n = 9$) outperformed the straight-forward implementation using conditionals by a factor of 2.5.

Texturing

By default, a pre-lit 2D texture is mapped onto the terrain. This can be a photo texture or, as for the Puget Sound, a synthesized 2D texture. During pre-processing, the texture is dyadically downsampled using a Lanczos filter with radius 2 to obtain a single, large mipmap. Now tiles are cut out of the mipmap to precisely match the tiles of our mesh hierarchy. To save GPU memory and bandwidth, each texture tile is then compressed using the S3 compression scheme. More specific, tiles are encoded using the DXT1 format, which yields good results for most photographic or synthetic textures at a compression rate of 6:1. We store the 16K^2 Puget Sound texture including 9 (11) mipmap levels for the 16K^2 (4K^2) geometry in about 170 MB.

If a pre-lit texture is not available, it is computed from the original terrain in a pre-process. Alternatively, normals could be stored as additional vertex attributes. However, besides the additional memory overhead that is introduced (at least two 8 bit values to cover the upper hemisphere), lighting artifacts due to non-continuous changes of normals during LOD transitions can only be resolved by storing one normal per vertex *and* level. On the other hand, a DXT1 pre-lit texture with 4 texels per vertex has the same storage requirements as a single per-vertex normal, but it avoids any lighting artifacts because texture filtering is performed *after* lighting.

6. MEMORY MANAGEMENT

After building the discrete LOD hierarchy, for high-resolution terrains including morph values and textures, the data is far too large to be stored in local video memory of recent GPUs. To avoid frequent paging of textures and vertex buffers, and to optimize progressive updates we have implemented a memory manager. At initialization time, the memory manager allocates chunks of exponentially growing sizes in GPU memory, to prevent external fragmentation. Sizes range from 32KB to a maximum size that allows the largest vertex buffer to be stored in such a chunk. Additionally a number of textures with a fixed resolution is allocated. The memory manager stores meta-data for each memory block, i.e. size, a time stamp, and the number of levels already sent to the GPU. Paging is now implemented as a mixture between a last recently used (LRU) and a tightest fit (TF) strategy.

Whenever a tile A is to be rendered, the system determines if there is already a chunk associated with A. If not, and also no appropriate chunk is available, the tile B with the earliest time stamp large enough to completely store A is determined. B is then marked as non-resident, and the chunk is overwritten with the data of A. To efficiently determine B, we keep a priority list for each available size. This allows us to weight the LRU strategy against a TF criterion. Once a chunk has been associated with A, all data required to render the current level is sent to the GPU. If there already was a chunk associated with A, the memory manager determines whether the chunk contains all necessary data. If not, the CPU sends all missing vertices and the

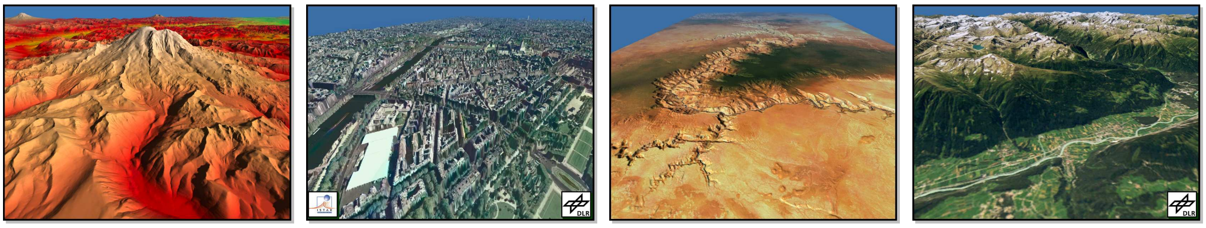


Figure 7: Test data sets in this paper. From left to right: Puget 16K×16K, Paris, Grand Canyon, and Alps. Observe the high degree of geometric details present even in regions further away from the viewer.

Data Set	Resolution	Texture	original Size	Storage	fps $\tau = 1$	MΔ/sec $\tau = 1$	fps $\tau = 5$
Puget4K	4K×4K	16K×16K	800MB	412MB	202	78.85	199
Puget16K	16K×16K	16K×16K	1.25GB	1.25GB	60	25.69	57
Grand Canyon	4K×2K	8K×4K	112MB	80MB	289	74.60	292
Paris	9.7K×5.8K	19.5K×11.7K	763MB	267MB	36	100.87	65
Alps	8.9K×8.5K	8.9K×8.5K	361MB	546MB	145	65.43	155

Table 1: Timings and Results. Original size only includes height field and texture, without taking mipmaps into account. τ refers to the pixel error. For $\tau = 1$ geomorphs were disabled, for $\tau = 5$ they were enabled.

required index buffer to the GPU. Since vertices are shared across levels, this update is usually very cheap compared to the upload of all vertices. Whenever a tile is rendered, its time stamp is updated.

The memory manager supports uniform load on the bus connecting the CPU and the GPU, thus avoiding 'paging hiccups': when a non-resident tile enters the view frustum, there is usually another one that has to be released, the texture tile has to be uploaded, and an initial LOD has to be sent to the GPU. However, with high probability this initial LOD requires only a few vertices. On the other hand, if a tile was already resident, performing an update only requires a fraction of the entire data to be sent.

Speculative prefetches are also supported, if there are unused memory chunks. If the number of chunks needed to render the current view falls below a certain fraction of all allocated chunks, the user's view is predicted. Whenever the user moves, a list containing the last viewing parameters is updated. By fitting a spline through these parameters, new viewing parameters can be extrapolated and tiles that are predicted to become visible in the near future can be prefetched, as long as a maximum time budget is not expired. In this way, very smooth fly-overs at high frame rates can be achieved.

7. RESULTS

Our results and timings are summarized in table 1. All timings were done on a P4 3.0GHz with 2GB RAM and GeForce 6800GT with 256MB. The machine was equipped with a single standard 120GB IDE hard disk. All data sets were rendered to a 1024×768 view port. Enabling 8x full-screen anti-aliasing and 4x anisotropic texture supersampling, the frame rate dropped about 30%. The timings should be fairly comparable to more recent publications. Though we have a newer graphics card, we render a considerably larger view port compared to many other systems.

Pre-processing of the geometry to a 9 level hierarchy processes approximately 15M vertices per minute and is linear in the amount of vertices. Memory consumption is constant, as tiles are processed independently of each other. Generating a 16K×16K texture hierarchy including filtering takes about 5 Minutes.

The Puget4K and the Grand Canyon data sets are only medium sized, and consequently our system is neither triangle nor memory limited. For the Paris data set with its 2.8MΔ per frame, we become triangle limited. Note however that this is a worst-case scenario, as our triangulation faithfully reconstructed all the steep sides of the buildings. A lot of these triangles are backfaces that are culled by OpenGL (but they are still counted since they pass the geometry stage). However, the Paris dataset is an excellent benchmark for the raw triangle throughput that our system can achieve.

The Puget16K dataset on the other hand is large enough to demonstrate the effects of the memory system. The lower triangle throughput reflects that our paging strategy does not come for free, but it still allows for highly interactive fly-overs

The Alps data set is a good mixture between these extremes. It contains lots of flat terrain around Munich and a considerable amount of very rough terrain around the Alps.

As our results show, frame rates for highly triangulated data sets, such as Paris, can also be improved by increasing the pixel error and enabling geomorphing. For these highly triangulated datasets we also hope to benefit from continuously increasing vertex processor throughput on future graphics chips.

8. CONCLUSION & FUTURE WORK

We have presented an efficient rendering system for large and textured terrain data that provides excellent quality and highly detailed views. In particular, at equal frame rates our system guarantees a smaller pixel error than previous approaches. We achieve these properties by exploiting a special discrete LOD hierarchy, as well as processing and rendering functionality on recent GPUs.

In the future, we will investigate dedicated compression schemes that are amenable to GPU decoding, such as vector quantization. Both, the possibility to compress mesh hierarchies as well as texture will be considered. As GPUs are become increasingly powerful, adaptive on-the-fly texture synthesis will become an important feature.

Acknowledgements

We would like to thank the DLR and ISTAR for the Paris and Alps data sets and the people from GA Tech [Geo] for making the Puget Sound and Grand Canyon datasets publicitly available.

9. REFERENCES

[Blo00] J. Blow. Terrain rendering at high levels of detail. In *Game Developer's Conference*, 2000.

[BSW83] R. E. Bank, A. H. Sherman, and A. Weiser. Refinement algorithms and data structures for regular local mesh refinement. In *Scientific Computing*, pages 3–17, 1983.

[CGG+03a] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, 2003.

[CGG+03b] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-sized batched dynamic adaptive meshes (p-bdam). In *Proc. IEEE Visualization '03*, pages 147–154, 2003.

[DWS+97] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In *Proc. IEEE Visualization '97*, pages 81–88, 1997.

[FEKR90] R. L. Ferguson, R. Economy, W. A. Kelly, and P. P. Ramos. Continuous terrain level of detail for visual simulation. In *IMAGE V Conference '90*, pages 144–151, 1990.

[FL79] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. In *Proc. ACM SIGGraph '79*, pages 199–207, 1979.

[Geo] Georgia Institute of Technology. Large Geometric Models Archive. http://www.cc.gatech.edu/projects/large_models.

[GGS95] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In *Proc. IEEE Visualization '95*, pages 135–142, 1995.

[GH95] M. Garland and P. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[HB87] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proc. ACM SIGGraph '87*, pages 103–110, 1987.

[HDJ04] L. M. Hwa, M. A. Duchaineau, and K. I. Joy. Adaptive 4-8 texture hierarchies. In *In Proc. Visualization*, pages 219–226, 2004.

[Hop98] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proc. IEEE Visualization '98*, pages 35–42, 1998.

[IBM] IBM Corp. T221 Flat Panel Monitor. http://www.ibm.com.

[KLR+95] D. Koller, P. Lindstrom, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Virtual GIS: A real-time 3D geographic information system. In *Proc. IEEE Visualization '95*, pages 94–100, 1995.

[LH04] F. Losasso and H. Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *Proc. ACM SIGGraph '04*, pages 769–776, 2004.

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proc. ACM SIGGraph '96*, pages 109–118, 1996.

[LP01] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proc. IEEE Visualization '01*, pages 363–370, 2001.

[LP02] P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[Mal00] H. S. Malvar. Fast progressive image coding without wavelets. In *Proc. IEEE Data Compression*, pages 243–252, 2000.

[Paj98] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *Proc. IEEE Visualization '98*, pages 19–26, 1998.

[PFL78] T. K. Peucker, R. J. Fowler, and J. J. Little. The triangulated irregular network. In *Proc. ASP-ACSM Symposium on DTM's*, 1978.

[Pom00] A. A. Pomeranz. ROAM using surface triangle clusters (RUSTiC). Master's thesis, Center for Image Processing and Integrated Computing, University of California, Davis, 2000.

[RHSS98] S. Röttger, W. Heidrich, P. Slusallek, and H. P. Seidel. Real-time generation of continuous levels of detail for height fields. In *Proc. WSCG '98*, pages 315–322, 1998.

[SN95] M. Suter and D. Nüesch. Automated generation of visual simulation databases using remote sensing and GIS. In *IEEE Visualization '95*, pages 86–93, 1995.

[Ulr00] T. Ulrich. Rendering massive terrains using chunked level of detail. ACM SIGGraph Course "Super-size it! Scaling up to Massive Virtual Worlds", 2000.

[Wil83] L. Williams. Pyramidal parametrics. In *Proc. ACM SIGGraph '83*, pages 1–11, 1983.

Improving Quality of Free-Viewpoint Image by Mesh Based 3D Shape Deformation

Satoshi Yaguchi †‡ Hideo Saito †

† Department of Information and Computer Science, Keio University
3-4-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522 JAPAN

‡NTT COMWARE Corporation, NTT Shinagawa TWINS Annex Bldg.
1-9-1 Konan, Minato-ku, Tokyo 108-8019 JAPAN
E-mail yagu@ozawa.ics.keio.ac.jp, saito@ozawa.ics.keio.ac.jp

ABSTRACT

In this paper, we present a method to synthesize high-quality virtual viewpoint image targeting the detailed texture objects. About 30 images are taken from multiple uncalibrated cameras around the object, and the Visual Hull model is reconstructed with Shape from Silhouette method. To deform 3D surface model that is converted from Visual Hull Model using the information such as image texture and object silhouette, the difference between the real object and the reconstructed model is evaluated as a cost function of optimization problem.

Our deforming model algorithm is based on single vertex iterative shifting. The vertex of surface triangle mesh is moved to the selected candidate point that maximizes the cost function. The cost function is consisted by four constraint criteria, texture correlation, smoothness, object silhouette, and mesh shape regularity. In addition to the cost function, such as judging mesh direction and combining / dividing meshes are applied for refined 3D models to avoid mesh folding and mesh size unevenness. The refined model provides a quite accurate dense corresponding relationship between the input images, so that high quality image can be synthesized at virtual viewpoint.

We also demonstrate the proposed method by showing virtual viewpoint images to applying the real image that are taken from multiple uncalibrated cameras.

Keywords

Shape-from-Silhouette, the Visual Hull, shape refinement, Image Based Rendering, weakly calibrated multiple camera system

1. INTRODUCTION

The acquisition of 3D geometric information and generating virtual viewpoint images from multiple cameras are studied many years and still researched actively. The studies of those methods are categorized into two basic methods, correlation-based stereo approach and the Visual Hull [Lau94a] model based approach.

The Multiple Baseline Stereo method [Oku93a, Nay98a] enables acquisition of 3D geometric information with correlation based stereo method from multiple cameras. The advantage of correlation-based approaches is able to handle concave regions. In contrast, it is not stable to handle occluded region. Occluded region causes losing pixels on synthesized virtual viewpoint images. Baselines between each camera are not short enough to remove occluded region, even if there are several tens cameras [Sai03a].

On the other hand, the Visual Hull model based approaches mainly reconstruct a 3D geometric information with shape from silhouette method [Mat00a]. These approaches can synthesize virtual viewpoint image without losing pixels even in partially occluded regions. However, the difference between the real object and reconstructed model may cause blurring in the synthesized virtual images. It is mainly caused by concave region or insufficient number of cameras to curve a Visual Hull model. Especially to apply an object with a detailed texture, the blur greatly effects image quality in appearance. Therefore the quality of the synthesized virtual

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

viewpoint image is degraded by inaccuracy of the reconstructed shape.

Recently, the advanced approaches that take into account the both advantages are intensively studied. Those approaches improve the image quality to refine 3D geometric shape with various information such as a texture correlation. The Space Curving Method [Kut00a, Sla02a] removes unnecessary voxels of the voxel-represented model using texture information by reducing difference between constructed models. The techniques for optimizing 3D model by deforming the vertex of surface triangle mesh based on the correlation of the texture have also been proposed [Eck04a, Nob03a].

We have been studied acquisition of 3D geometric information and generating virtual viewpoint images from multiple cameras. We have already proposed a new framework for the Visual Hull based virtual viewpoint image synthesizing method expanding to weakly calibrated multiple cameras using the "Projective Grid Space (PGS)"[Sai99a] in which the coordinates are defined by epipolar geometry between cameras instead of strong calibration [Yag02a, Yag04a]. In the framework, virtual viewpoint images can be synthesized with Image Based Rendering using correspondence map derived from the model like a view morphing method [Sei96a]. However, there is still remained the blurring problem, because the difference between the reconstructed model and the real object is not well considered.

The goal of this paper is to improve a quality of virtual viewpoint image that is synthesized with the framework. The proposed method in this paper aims to reduce the blurring effect to deform the reconstructed model with the information such as image texture or object silhouette. Quite accurate dense correspondence map between the images that is derived from the refined model enables synthesizing high quality virtual viewpoint images without the blur.

2. PROPOSED METHOD

The approach of the proposed method is based on the "Projective Grid Space (PGS)" framework [Sai99a], which relates the 3D object space with 2D image coordinates. Using PGS framework, reconstructing 3D shape model with Shape from Silhouette method and generating virtual viewpoint images becomes possible from uncalibrated multiple cameras [Yag04a].

Input images are taken by uncalibrated cameras that observe around a target object, and silhouette images are synthesized from those images. Several correspondence points between each image are extracted from feature points, and fundamental matrices, which are used for relating 3D object space and 2D image plane with the PGS framework, are calculated from those points. (Please refer the *appendix* or those papers, the details of the PGS framework is shown.)

An initial voxel model is reconstructed by the shape-from-silhouette method on the PGS. By applying the Marching Cubes algorithm, the Visual Hull model is converted to the surface representation model. The vertex of surface triangle mesh is moved to the selected candidate point maximizes the cost function that is consisted by four constraint criteria.

The cost function can be computed only on 2D image domain according to the projection computation from PGS to every image. Obtaining a dense correspondence map between images from the optimized model, high quality virtual view images are synthesized as the image interpolating two input images.

3. SHAPE REFINEMENT

This section describes proposed refinement technique of the 3D-surface model by using only 2D image domain information. Model shape is refined by moving each vertex of surface triangle mesh independently. Each vertex of the surface model is visited sequentially, and then cost function is evaluated at initial position of the vertex. If the evaluated value is under threshold, candidate points of refinement vertex are defined, and each cost value is calculated at those candidate points respectively. The vertex is moved to the candidate point that maximizes the cost function. When all vertexes are visited *N* times, or cost function is over threshold at all vertexes, shape refinement is finished.

3.1 Vertex Position Optimization

Our algorithm of deforming 3D shape is based on shifting a single vertex iteratively. Each vertex of reconstructed model is visited respectively and shifted independently. The process of shifting vertex position is performed by selecting and moving the candidate vertex to the point that maximizes cost function. The candidate vertexes are defined every iterating cycle.

To simplify the algorithm, the candidate points are defined on the line passing through the target vertex v_0 and a point *g* that is defined as the center of all adjacent vertexes x_0 , x_1 , ..., x_m . Then $2n+1$ candidate points v_n , ..., v_{-n} , (within the target vertex) are defined outside and inside of the model surface at intervals of a unit vector scaled by a weight *s* (see Figure 1).

The parameter s is decided dynamically depending on the value of the cost function of v_0 to enable detailed search, and that is ranged $0.1 \leq s \leq 1.0$. The number of candidate point n is decided depending on the distance between v_0 and g to preserve initial shape detailed structures.

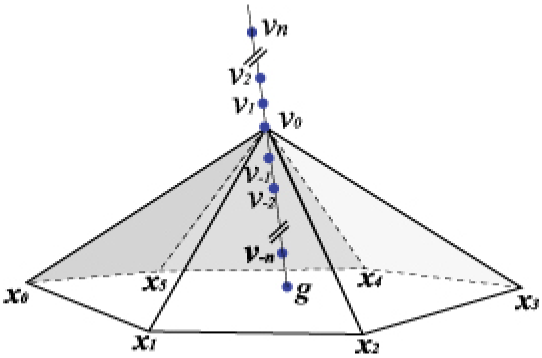


Figure 1. Deform of vertex: Moving candidate points are defined on a line v_0 to *g*.

Cost function V_{-n} , V_{-n+1} , V_0 , …, V_n of each candidate point v_n is calculated by the following equation (1).

$$\begin{aligned} V_n = &\alpha \cdot V_{corr}(v_n) \\ &+ \beta \cdot V_{smooth}(v_n) \\ &+ \lambda \cdot V_{shape}(v_n) \qquad (1) \\ &+ \delta \cdot V_{sil}(v_n) \end{aligned}$$

V_{corr} : Texture correlation
V_{smooth} : Smoothness
V_{shape} : Triangular shape regularity
V_{sil} : Silhouette

where α , β , γ , δ , ε are weighting coefficients. Definition of each criterion is described next.

3.1.1 Texture Correlation

Texture correlation of a vertex is determined by the texture of its adjacent triangle meshes. The correlations between each image are calculated by normalized cross correlation to apply all pixels of the mesh. The correlations are defined for the input images from which the vertex and the adjacent meshes are able to observe.

Vertexes of triangle mesh v_{0i} , x_{0i} , x_{1i} in the image *i*, and v_{0j} , x_{0j} , x_{1j} in the image *j* are related to 3D position by fundamental matrices. Inner point *p* of triangle mesh in the image *i* is corresponded to the point in the image *j* by affine transform A_{ij} (see Figure 2).

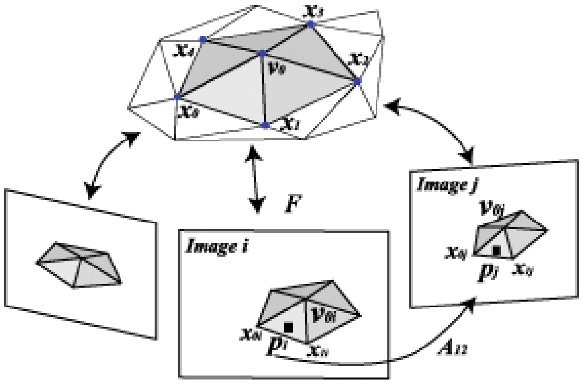


Figure 2. Texture correlation: Adjacent triangle Meshes are projected on images by fundamental matrixes. Each pixel of mesh is corresponded other images by Affine transformation matrixes.

The correlation of v_n of adjacent mesh k between image *i* and image *j* is shown equation (2).

$$V_{corr_{kij}}(v_n) = \frac{\sum_p (W_{p_i} - w_i)(W_{p_j} - w_j)}{\sqrt{\sum_p (W_{p_i} - w_i)^2 \sum_p (W_{p_j} - w_j)^2}}$$

(2)

where W_{p_i} and W_{p_j} are the color of the point p_i and p_j , respectively, w_i , w_j are the averages of color of all pixel in the image *i* and *j*, respectively.

$V_{corr_{kij}}(v_n)$ is also calculated for all combination of pairs of images on which the vertex v_n is projected onto the pair images without occlusion. Then total $V_{corr}(v_n)$ is computed as the average of all $V_{corr_{kij}}$ calculated for the vertex v_n . $V_{corr_{kij}}(v_n)$ is ranged $-1 < V_{corr}(v_n) < 1$.

3.1.2 Smoothness Constraint

Though surface of the object should be locally smooth, and be continuous, we apply to following smoothness constraint. The constraint is defined depending on the distance $d(v_n)$ between the vertex v_n and g, which is determined by all adjacent vertexes of the target vertex as described in **3.1.** In order to reduce over-smoothing, the distance $d(v_0)$ between g and the point v_0 , which is the 1/6 distance point on the line segment between v_0 and *g*, is subtracted from $d(v_n)$. Thus, we apply to following function as the smoothness constraint.

$$V_{smooth}(v_n) = \left(d(v_n) - d(v_0)/6\right)^2 \qquad (4)$$

where d is the distance between v_n and *g*, and weighting coefficient β will be negative.

 ISBN 1213-6972 ISBN 80-86943-09-7

3.1.3 Silhouette Constraint

The 3D-surface model after deformed should be filled over the initial Visual Hull sufficiently. The vertexes that determine the Visual Hull are constrained to the boundary of the initial silhouettes. However, because the Visual Hull model doesn't accurately express the real contour, for instance, the concave regions, the vertex projected on the boundary of the silhouette is only a candidate determining the contour. The silhouette constraint keeps the refined model to form the initial silhouette.

Therefore the following silhouette constraint is applied to the target vertex and the refinement candidate points (see Figure3).

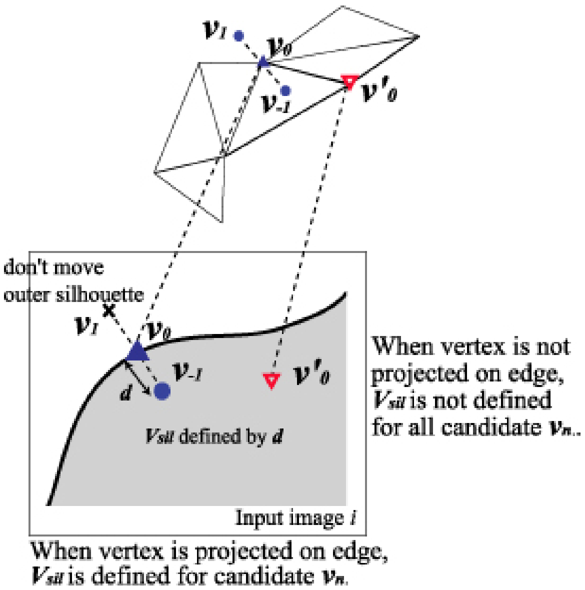


Figure 3. Silhouette Constraint: An image in that vertex projected on edge, silhouette constraint criterion is defined depending on the distance from edge.

1. When the target vertex v_0 is not projected onto the boundary of the silhouette in input image i, $V_{sil\ i}$ is not defined ($V_{sil}(v_n)$ of all candidate points are decided to 0.).

2. When v_0 is projected onto the boundary of the silhouette on input image i, $V_{sil\ i}$ is defined in proportion to the distance between the projected coordinate of candidate points v_n and that of v_0.

In addition,

3. If v_n is projected on outer silhouette even by one input image, v_n is excluded from candidate.

After all, $V_{sil}(v_n)$ of the candidate point is defined by the sum of $V_{sil\ i}(v_n)$ as following equation (5).

$$V_{sil}(v_n) = \sum_i V_{sil\ i}(v_n) = \sum_i d_i \quad (5)$$

3.1.4 Constraint on Triangular shape regularity

As iterative vertex deforming process goes on, mesh shape and size are changed. If triangle of mesh is very small or very narrow, its texture is not able to derive enough to calculate correlation. So it is preferable that triangle shape is kept shape regularity and size as same as possible.

Therefore, the criterion of constraints on the triangular shape regularity $V_{shape}(v_n)$ of adjacent mesh k of vertex v_n is shown by equation (6) [Eck04a].

$$V_{shape\ k}(v_n) = 2\sqrt{3} \cdot \frac{|\vec{e}_1 \times \vec{e}_2|}{|\vec{e}_1|^2 + |\vec{e}_2|^2 + |\vec{e}_3|^2},$$

$$0 < V_{shape} \le 1 \quad (6)$$

where $\vec{e}_1$, $\vec{e}_2$, $\vec{e}_3$ are edge vectors. This function represents the geometric quality of triangle area rations

The criterion $V_{shape}(v_n)$ of vertex v_n is defined the average of $V_{shape\ k}(v_n)$.

3.2 Additional Constraints

Going on the iterating process, vertex shift may cause mesh folding or mesh size unevenness. Mesh folding changes the topology of the model or the visibility of the vertex, the correlation of the mesh is not able to calculate accurately and the mesh is not rendered correctly (such as "hole" or "overlap" texture is appeared in the synthesized image).

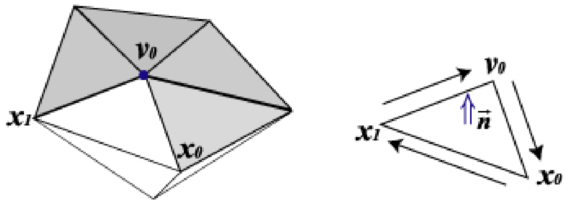
Mesh size variation causes inequality of the precision of cost function, and very small meshes cause invalidate vertex because of having not enough areas deriving textures. In order to avoid such folding or unevenness of the mesh size, following additional constraints are also considered.

3.2.1 Avoiding Mesh Folding

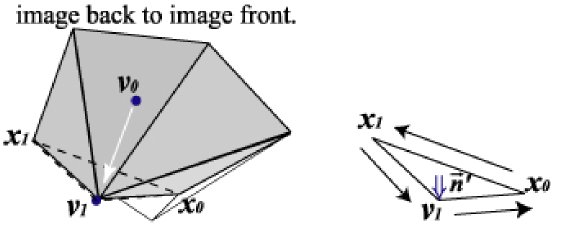
Mesh folding is occurred when the mesh is not occluded turns to be occluded. It is stated differently, it is caused when the vertex is moved over the boundary of the adjacent mesh.

To avoid such mesh folding, normal direction of mesh on the projected image can be used. Vertexes of mesh v_0, x_1, x_2 are indexed in the order of clockwise on the projected image, if the mesh is not occluded, as shown in Figure 4(a). Then the direction of surface normal of the all the meshes adjacent to v_0 are calculated with edge vector. According to the direction, every mesh is determined if it is on the right side or wrong side.

If one of the meshes is not on the right side, mesh is considered as folding as shown in Figure 4(b). Such candidate points of v_0 are excluded from the vertex searching candidate points that are defined **3.1**.



(a)Initial position: Surface normal vector $\vec{n}$ directs through image back to image front.



(b)Mesh folding : The point that changes direction of $\vec{n}$ is excluded out the vertex shift candidate.

Figure 4. Mesh folding: If the candidate point is decided over the boundary of the adjacent mesh, rotation of vertexes and surface normal vector are changed in reverse.

3.2.2 Merging / Dividing Meshes and Vertexes

The distance between the vertexes is changed after the iterating process. Constraint on triangular shape regularity described **3.1.5** is not able to avoid mesh size unevenness. So, when every iteration cycle is finished, meshes and vertexes are divided or merged according to the distance between adjacent vertexes.

If the distance between adjacent vertexes is over the threshold distance D_{max}, new vertex is inserted to the middle point in these vertexes, and triangles that sharing these vertexes are divide into two triangles each. If the distance between each adjacent vertex is less than the threshold distance D_{min}, these vertexes are merged into one vertex. When two vertexes should be merged, first, the vertex after merged is decided for the middle point of these vertexes. Next, two triangles that share both vertexes are removed, and the mesh index references of each vertex are re-indexed to refer the new vertex.

4. EXPERIMENTAL RESULT

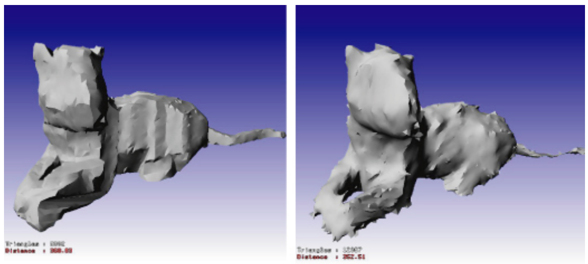
The proposed method has been tested with several real objects. Input images are taken by uncalibrated camera as color images (640×480 pixels BMP format). Results for two real objects, a Jaguar and an elephant are shown in this section.

4.1 Jaguar

The target object is a paper craft of "Jaguar" about 20cm×10cm×10cm. 36 images were taken around a target object with a hand-held camera as input images.

(1) input image 1 (2) input image 2

Figure 5. Real Images from that virtual view image shown Figure 7 are synthesized to interpolate.



(1)Initial Model (2)Refined Model (N=300)

Figure 6. Initial surface model and refined model iterated N=300.

Figure 5 shows the example of the input image. An initial surface model reconstructed in the PGS and the refined model with iteration N=300 by proposed method are shown in Figure 6.

Virtual viewpoint image were synthesized with interpolation ratio 5:5 of two real images as shown in Figure 7. Here, Figure (1) and (2) represent the image synthesized from initial model, and the image synthesized from refined model iterated N=300, respectively.

In the image of initial model (Figure 7 (1)), the surface texture of the jaguar is blurred. The background area image (blue area) is also rendered as the surface texture of the object. Those bad rendering effects are caused by inaccurate shape of the initial model. Especially in the white ellipse area, the discontinuous pattern is seen, that is because of wrong shape of the jaguar's foot. On the other hand, such bad rendering effect is completely removed in refined model image (Figure 7 (2)).

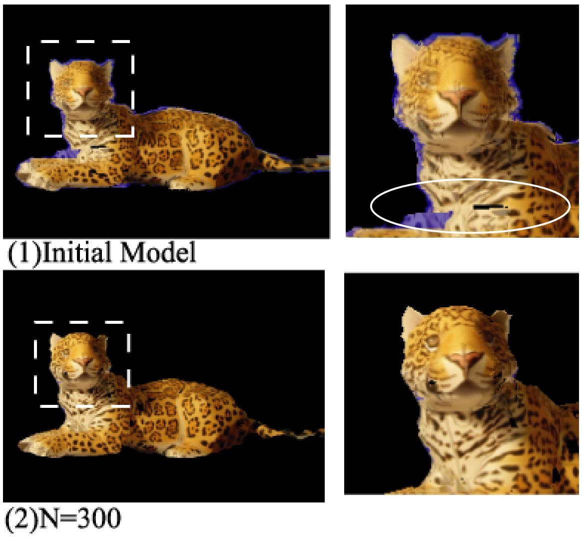


Figure 7. Virtual view image

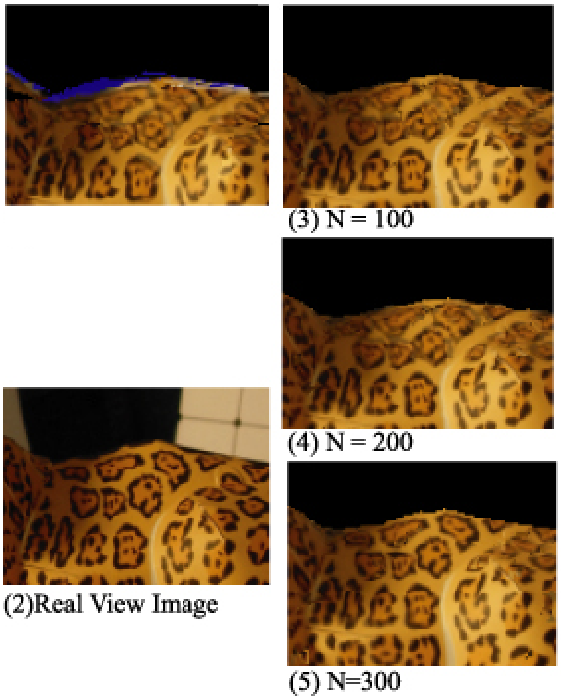


Figure 8. comparison of textures: (1) Rendering from initial 3D shape model. (2) Real image texture about the same viewpoint as virtual viewpoint. (3), (4), and (5) Rendered images with refined model of iteration *N*=100, *N*=200, and *N*=300, respectively.

Next, Figure 8 shows the comparison of texture for each iterating number, initial model, *N*=100, *N*=200, *N*=300, and real viewpoint image. It is understood that the image quality of synthesized image has been improved as the number of iterating process is increased. The real image shown in Figure 8 (2) is taken from about same viewpoint as virtual viewpoint. By comparing *N*=300 image (Figure 8 (5)) with real viewpoint image, about the same quality texture is acquired with refined model.

4.2 Elephant

The target object is an "elephant" about 20cm×20cm×20cm. 30 images were taken as input images. The examples of input images are shown in Figure9.

Figure 10 shows an initial surface model and the refined model with iteration *N*=450 by proposed method. Figure 11 shows the synthesized free viewpoint images. Images on either side are reference real images and middle images are synthesized changing interpolation-weighting factor. Upper images were synthesized from initial model, and lower were synthesized from refined model iterated with *N*=450. In the upper images, many background image areas (blur areas) are distributed, and the border of adjacent rectangular patches are blurred. On the other hand, such bad rendering effects are reduced in the images rendered with the refined model. In this way, free viewpoint images with improved quality texture as same as input images are able to synthesized using proposed method.

Figure 9. Examples of input images.

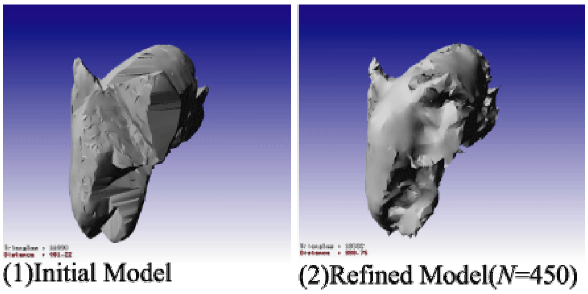


Figure 10. Initial surface model and refined model iterated *N*=450.

5. CONCLUSION

We proposed a method for improving the quality of new view image by deforming 3D surface model from uncalibrated multiple cameras. The cost function of deforming 3D shape model is defined only on 2D image domain according to the projection computation from PGS to every image. Deforming 3D shape model with the texture correlation and other constraints reduces the difference between the reconstructed model and the real object.

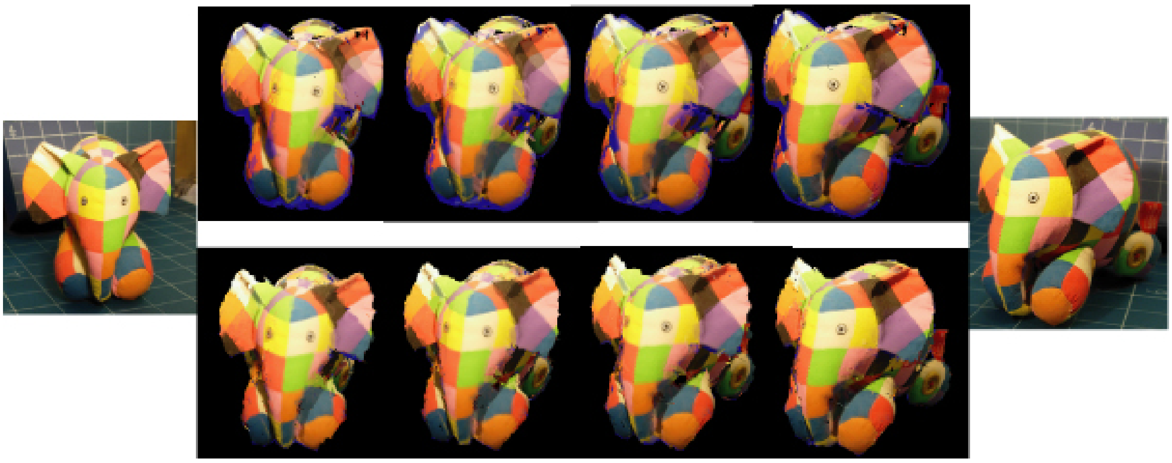


Figure 11. Synthesized free viewpoint images. Images on either side are reference real images, the middle upper images were synthesized from initial model, and lower were synthesized from refined model iterated with *N*=450.

Using refined model removes the blur on the image texture and enables synthesizing high quality free viewpoint image. Our method requires only about 20 to 30 images and does not require strong calibration, so that it is easy to use our method with simple camera systems. For the future work, we will extend this proposed method to the dynamic events and synthesized free viewpoints video from uncalibrated simple camera systems.

6. REFERENCES

[Eck04a] Eckert, G., Wingbermuhle, J. and Niem, W. Mesh Based Shape Refinement for Reconstructing 3D-Objects from Multiple Images: The First European Conference on Visual Media Production (CVMP04), Mar.2004.

[Che93a] Chein, S. and Williams, L. View interpolation for image synthesis: Proc. SIGGRAPH '93, pp.279-288, 1993.

[Cur96a] Curless, B. and Levoy, M. A Volumetric Method for Building Complex Models from Range Images: Proc.of SIGGRAPH '96, 1996.

[Kut00a] Kutulakos, K.N. and Seitz, S.M. A Theory of Shape by Space Carving: IJCV(38), No.3, pp.199-218, 2000.

[Lau94a] Laurentini, A. The Visual Hul Concept for Silhouette-based image understanding: IEEE Trans Pattern Andl. Machine Intell., 16(2), pp150-pp162, Feb. 1994.

[Nay98a] Narayanan, P.J., Rander, P.W. and Kanade, T. Constructing Virtual Worlds using Dense Stereo: Proc. ICCV 98, 1998.

[Mat00a] Matusik, W., Buehler, C., Raskar, R., Gorlter, S. and McMillan, L. Image-Based Visual Hulls: Proc. of SIGGRAPH 2000, 2000.

[Nob03a] Nobuhara, S. and Matsuyama, T. Dynamic 3D Shape from Multi-Viewpoint Images using Deformable Mesh Models: Proc. of 3rd International Symposium on Image and Signal Processing and Analysis, Rome, Italy, September 18-20, pp. 192-197, 2003.

[Oku93a] Okutomi, M. and Kanade, T. A Multiple-Baseline Stereo: IEEE Trans. on PAMI, Vol.15, No.4, pp.353-363, 1993

[Sai99a] Saito, H. and Kanade, T. Shape Reconstruction in Projective Grid Space from Large Number of Images: IEEE Proc. Computer Vision and Pattern Recognition, Vol. 2, pp. 49-54, 1999.

[Sai03a] Saito, H., Baba, S., Kanade, T. Appearance-Based Virtual View Generation From Multicamera Videos Captured in the 3-D Room: IEEE Trans. on Multimedia, vol.5, no.3, pp. 303-316, Sep. 2003

[Sei96a] Seitz, S. M. and Dyer, C. R. View Morphing: proc. of SIGGRAPH '96, pp. 21-30, 1996.

[Sla02a] Slabaugh, G.G., Schafer, R.W. and Hans, M.C. Multi-Resolution Space Carving Using Level Set Methods: Proc.ICIP02, Vol.II, pp.545-548, 2002.

[Yag02a] Yaguchi, S. and Saito, H. Arbitrary Viewpoint Video Synthesis from Uncalibrated Multiple Cameras: WSCG'2002 - the 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2002, Feb.2002

[Yag04a] Yaguchi, S. and Saito, H. Arbitrary Viewpoint Video Synthesis from Multiple Uncalibrated Cameras: IEEE Trans. on Systems, Man and Cybernetics, PartB, vol. 34, no1, PP.430-439, 2004.

APPENDIX

In our method, 3D point is related to 2D image point without estimating the projection matrices by "Projective Grid Space (PGS)", which can be determined by only fundamental matrices representing the epipoler geometry between two basis cameras. Because the 3D coordinate of PGS is dependently defined by the camera image coordinates, 3D position of any sample points does not have to be measured.

Projective Grid Space

The "Projective Grid Space (PGS)" is defined by camera coordinate of the two basis cameras. Each pixel point (*p*, *q*) in the first basis camera image defines one grid line in the space. On the grid line, grid node points are defined by horizontal position *r* in the second image.

Thus, the coordinate *P* and *Q* of PGS is decided by the horizontal coordinate and the vertical coordinate of the first basis image, and the coordinate *R* of PGS is decided by the horizontal coordinate of the second basis image. Since fundamental matrix F_{21} limits the position in the second basis view on the epipolar line *l*, *r* is sufficient for defining the grid point. In this way, the projective grid space can be defined by two basis view images, of which node points are represented by (*p*, *q*, *r*).

3D-2D Mapping

As described in the previous section, the PGS is defined by two basis views, and the point in the PGS is represented as *A*(*p*, *q*, *r*). The point *A*(*p*,*q*,*r*) is projected onto a_1 (*p*, *q*) and a_2 (*r*, *Y*) in the first basis image and the second basis image, respectively. The point a_1 is projected as the epipolar line *l* on the second basis view expressed as equation (7).

$$l = F_{21} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix} \quad (7)$$

where F_{21} represents the fundamental matrix between the first and second basis images. The point a_2 is onto *l*, thus the coordinate of a_2 (*r*, *Y*) is decided (see Figure 12).

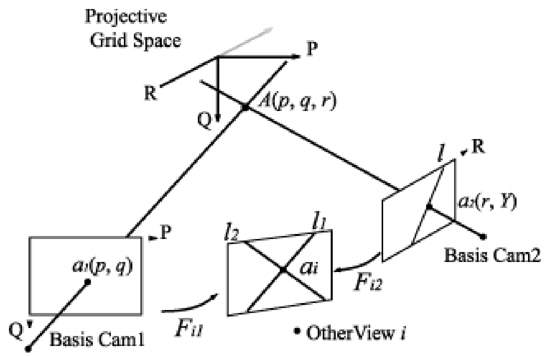


Figure 12. Projection of 3D point onto an image: The point on the projective grid space is projected to the cross point of two epipolar lines in the image of view *i*.

The projected point in *i* th arbitrary real image is determined two fundamental matrices, F_{i1} , F_{i2} between two basis images and *i* th image. The projected point in the *i* th image must be on the epipolar line l_1 of a_1 in the first basis image, which is derived by the F_{i1} . In the same way, the projected point in the *i* th image must be on the epipolar line l_2 of a_2 , which is derived by the F_{i2} .The intersection point between the epipolar line l_1 and l_2 is the projected point *A*(*p*, *q*, *r*) in the *i* th image. In this way, every point of the PGS are projected onto every image, where the relationship can be represented by only the fundamental matrices between the image and two basis images.

In this way in the PGS, the fundamental matrices between images determines 3D-2D mapping. In the case of reconstructing 3D shape model with the shape-from-silhouette method, the voxels within a certain region of the PGS are projected into the silhouette image. And the point in an image is able to correspond to other image points to use correspondence map derived from the reconstructed model. And so the correspondence map enables to synthesize new view images as interpolated views and enables to calculate texture correlation between input images.

Real Time Rendering of Atmospheric Scattering and Volumetric Shadows

Biri Venceslas
Charles Cros Institute
6 bd du Danube
F-77700 SERRIS
FRANCE
biri@univ-mlv.fr

Arquès Didier
Charles Cros Institute
6 bd du Danube
F-77700 SERRIS
FRANCE
arques@univ-mlv.fr

Michelin Sylvain
Charles Cros Institute
6 bd du Danube
F-77700 SERRIS
FRANCE
michelin@univ-mlv.fr

ABSTRACT

Real time rendering of atmospheric light scattering is one of the most difficult lighting effect to achieve in computer graphics. This paper presents a new real time method which renders these effects including volumetric shadows, which provides a great performance improvement over previous methods. Using an analytical expression of the light transport equation we are able to render directly the contribution of the participating medium on any surface. The rendering of shadow planes, sorted with a spatial coherence technique, and in the same philosophy than the shadow volume algorithm will add the volumetric shadows. Realistic images can be produced in real time for usual graphic scenes and at a high level framerate for complex scenes, allowing animation of lights, objects or even participating media. The method proposed in this paper use neither precomputation depending on light positions, nor texture memory.

Keywords : Real time rendering / Volumetric shadows / Single scattering / Participating media

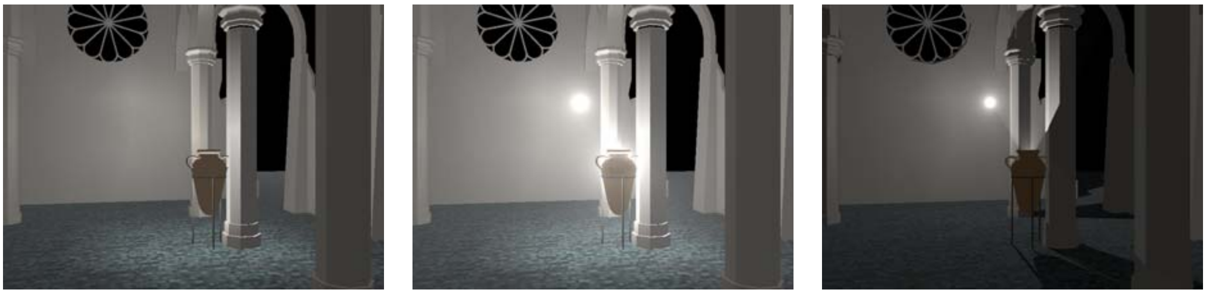


Figure 1: The same scene lit a. (left) classically, b. (center) with single scattering and c. (right) with single scattering and volumetric shadows (right) .

1. INTRODUCTION

The growing capacities of graphic cards enable the rendering of more and more complex physical models in real time, like anisotropic reflection or environment mapping. Therefore, it is not a surprise if a current challenge in computer graphics is the accurate rendering of atmospheric effects, and especially the light scattering. Atmospheric light scattering is due to little particles, like dust or water, that lay in the air, scattering and absorbing the light they receive. They creates effects such light beams, shafts of light and visibility loss. These phenomena often occur under foggy or smoky conditions but are also visible by clear or cloudy weather in the presence of sunlight.

Unfortunately, rendering such lighting effects in real time remains quite complex since they depend on camera and light positions and since they occur everywhere in the space. Introducing such effect in traditional graphic engine will greatly enhance the

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

realism of the virtual scene and have many applications [Ru94]. Considering the particular situation of figure 1, it is clear that rendering the participating medium is not enough. Here, the representation of shadow volumes is necessary to obtain a realistic image. Thus there is a need for a simple algorithm, easily integrated in traditional algorithms, able to render those effects.

In this paper, we present a new algorithm that fulfills this goal. It can render accurately participating media, including effects like light beams in foggy or smoky scenes, or any other atmospheric scattering effects. The participating media can be isotropic or anisotropic and are lit by one or several, static or moving, point light sources since no precomputation are done involving either lights or camera. Our technique produces high resolution images and takes into account volumetric shadows, cast by occluders contained in the media. Without any texture memory cost, but using intensively graphics hardware, our method can render images at a high frame rate, and is real time for classical graphic scene. The method is also easy to implement in traditional graphic engines since it follows the same strategy than the shadow volume algorithm. Therefore, it is straightforward to obtain animations where objects, sources and even participating media can move.

2. PREVIOUS WORK

The representation of participating media has been a real challenge for years and the literature about it is abundant. We can easily divide all these studies between the single scattering methods and the multiple scattering ones. Multiple scattering methods try to compute all light reflections and inter-reflections inside a medium, whatever the number of these ones. This complex situation is difficult to handle but is essential in the rendering of clouds for example. Multiple scattering illumination can be obtained by determinist methods [RT87, Ma94, ND96] or by stochastic methods [PM93, LW96, JC98] and sometimes involve a resolution of the flow equations like in [FM97, St99, DK00, FS01]. Despite their realism, they suffer from excessive computation times due to the complexity of light exchanges occurred in these cases. Therefore it is not suitable for our goal and we will focus on single scattering methods.

These techniques [NM87, Ma94, DY00, HP02, DY02] approximate the multiple reflections of light as a constant ambient term and consider only the first scattering of light ray in the direction of camera. This assumption allows a direct rendering of the illumination of the medium which is more suitable for interactive rendering. Visualization is often done by ray tracing or ray marching. View rays are followed to gather the participating media contributions. Unfortunately, these methods [FM97, JC98], are far from being real time on a conventional desktop computer. With the growing capacities of graphics hardware, the real time problem has been investigated.

Two approaches can be used to achieve this goal: volume rendering or direct representation. To add the volumetric shadows the first approach will use naturally shadow maps techniques when the second one is oriented to shadow volumes algorithm [He91]. Volume rendering is a classic solution to render participating medium which is a volume *de facto*. Methods like [BR98, WE98, St99, FS01, NM01] represent densities or illumination in voxels encoded into 2D or 3D textures. Accumulation techniques using textured slices or virtual planes are then used to display the result. That kind of methods could produce nice images of clouds or gas. But apart from requiring a lot of texture memory, they are not suitable for shafts of light where sharp edges exist. Special methods are defined to render beams and shafts of light precisely and most of them [DK00, DY00, Ev02, LG02] use volume rendering techniques along with sampling shadows in shadow maps. But they suffer from artifacts due to the sampling. Dobashi et al. [DY02] presents a very elegant solution to solve this problem using specialized adaptive sampling for shadows. They obtain an interactive rendering of participating media without aliasing or artifacts. However the image resolution remains small since the method is expensive in terms of fillrate. Moreover, the method works only with static lights due to the precomputation of shadow maps.

The algorithms belonging to the second approach computes directly, on every point in the scene, the contribution of the participating medium. This is well adapted to classical graphic engines since it consists in one more rendering of the scene. In this case, methods like [Me01, HP02] use participating medium boundaries, or special virtual planes, combined with vertex and fragments shaders. Other methods focus on the rendering of the atmosphere [On05]. A last method of this group is proposed by Sun et al. [SR05] and is the only one to consider the effect of light scattering on the illumination of objects. Despite it is real time, it does not take into account shadows. Our work belongs also to this group and is the only one of them to integrate realistic lighting effect with volumetric shadows.

3. OVERVIEW OF OUR METHOD

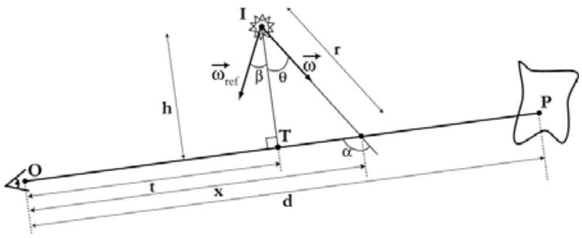
To obtain real time performances, we consider only one scattering of light in the medium. Multiple scattering is approximated by a constant ambient term in the scene and each participating medium is homogeneous.

The algorithm exploits an analytical expression of the total contribution of scattered light along a view ray. This allows the direct computation of this contribution between the camera and any point of the scene. Therefore, we compute the previous expression:

- on scene vertices or on boundaries of participating media.
- on any point of the shadow planes.

As stated before, our method is close to shadow volume techniques [He91] or other algorithms using shadow planes [AA03]. Indeed, after having compute and render light scattering contribution of lit areas, we do the same for the shadow planes of any object that is set to cast shadows.

These shadow planes are classically obtained by using the object silhouettes regarding to the point light position and meshed. After a back to front sorting of the shadow planes, we render them. The participating medium contribution will be added if the shadow plane is frontfacing and subtracted if backfacing to take into account the volumetric shadows.



k_t	: Extinction coefficient	Is	: Light intensity
Ω	: Albedo	p	: phase function
L	: Luminance		

Figure 2 : Single scattering case and notations.

4. THEORETICAL BACKGROUND

As light is progressing through a participating medium, it have four interactions with it : absorption, emission, scattering and in-scattering [SH92]. If we consider only single scattering, the luminance of a point P seen from a point O can be written [SH92] :

$$L(O) = L(P)e^{-k_t d} + \frac{k_t \Omega}{4\pi} \int_0^d \frac{I_s(\vec{\omega}) e^{-k_t x} e^{-k_t r}}{r^2} p(\alpha) dx \qquad (1)$$

x, r, d, h are geometrical factors (cf. figure 2 for notations), while k_t is the extinction coefficient, and Ω is the albedo. I_s is the directional intensity of the source. The first term takes into account the scattering and the absorption while the second one is the in-scattering which is responsible for the subtle effects of atmospheric scattering. This equation is called the integral transfer equation.

4.1 Angular formulation of the integral transfer equation

The integral transfer equation can be written [LM00] using the angle between the view ray and the direction toward point light. This formulation will be used to obtain an analytical solution of the previous equation. Indeed, instead of integrating the integral transfer equation regarding to the distance x along the ray, we choose to use the variation of the angle θ between the vector $\vec{\omega}$ and the vector $\overrightarrow{ST}$ defined by the orthogonal projection of the point light source on the view ray. Using this variable change :

$$r = \sqrt{h^2 + (x - t)^2}$$

we can obtain (see [LM00]) :

$$L(O) = L(P)e^{-k_t d} + L_m(P)$$

where :

$$L_m(P) = \frac{k_t e^{-k_t t} \Omega}{4\pi h} \int_{\theta_0}^{\theta_d} I_s(\theta + \beta) e^{-k_t h \frac{\sin(\theta)+1}{\cos(\theta)}} p(\theta + \frac{\pi}{2}) d\theta$$

Further on, $L_m(P)$ will be called the medium contribution of the point P.

The kernel Λ of the previous integral is complicated enough to prevent any analytic integration. But we can approximate this kernel to obtain a much more simple expression. The function in the kernel without considering the light intensity, depends only on the angle θ, considering that the extinction coefficient is constant, i.e. that the participating medium is homogeneous. Therefore, we can develop its expression in a polynomial base (we use 4 degree) :

$$e^{-k_t h \frac{\sin(\theta)+1}{\cos(\theta)}} p(\theta + \frac{\pi}{2}) \approx c_0(k_t, h) + c_1(k_t, h)\theta + ...$$

For traditional phase functions – isotropic, hazy, murky, etc. – the formal expressions of coefficients c can be obtained in the annex. We introduce this equation in the expression of $L_m(P)$ to obtain :

$$L_m(P) = \frac{k_t e^{-k_t t} \Omega}{4\pi h} \left[c_0(k_t, h) \int_{\theta_0}^{\varsigma_d} I_s(\theta + \beta) d\theta + c_1(k_t, h) \int_{\theta_0}^{\varsigma_d} I_s(\theta + \beta)\theta d\theta + ... \right]$$

Finally, for non directional point light source, these integrals are easily computed :

$$L_m(P) = \frac{k_t e^{-k_t t} \Omega}{4\pi h} \left[c_0(k_t, h) [\theta]_{\theta_0}^{\theta_d} + c_1(k_t, h) \left[\frac{\theta^2}{2} \right]_{\theta_0}^{\theta_d} + ... \right] \qquad (2)$$

 ISBN 1213-6972 ISBN 80-86943-09-7

and so we can obtain the single scattering contribution created by a point light source along any view ray in constant time.

A study on the quality of these approximations can be found in [Le01]. In general, they are quite good except when the ray passes close to the source, or when the observer is far from the source. In the first case, the contribution is so high, and in the second case, so small, that these errors remain unnoticeable.

Based on equation (2), we are now able to compute in "constant time" – i.e. without any numerical integration – the contribution of in-scattering light along a ray contained in a participating medium.

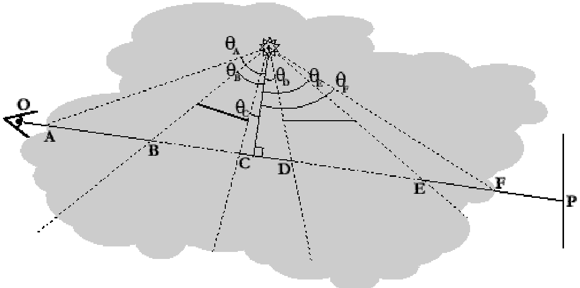


Figure 3 : A view ray partially in shadows.

4.2 Considering shadow volumes

Previous equations describe the particular case where the view ray remains totally lit and lays in the participating medium. To integrate shadow volumes and bounded participating medium, we need to consider more general cases, illustrated in figure 3. Indeed, due to shadows, the part of the ray laying in the medium could be split into lit and shadowed parts. In this example, the medium contribution along the ray is split into three parts on AB, CD and EF. The contribution of the single scattering of the ray OP is then:

$$L_m(P) = \frac{k_t e^{-k_t t}\Omega}{4\pi h}\left[\int_{\theta_A}^{\theta_B}\Lambda(\theta)d\theta + \int_{\theta_C}^{\theta_D}\Lambda(\theta)d\theta + \int_{\theta_E}^{\theta_F}\Lambda(\theta)d\theta\right]$$

The key idea of our approach is to rewrite this equation into a sum of differences. Indeed the light contribution of segment EF for example can be seen as the contribution of segment OF minus the one from segment OE. If we denote $\Gamma_m(P)$ the expression (2) for a lit ray between the camera center O and any point P, then the previous equation can be written :

$$L_m(P) = \left[(\Gamma_m(F) - \Gamma_m(E)) + (\Gamma_m(D) - \Gamma_m(C)) + (\Gamma_m(B) - \Gamma_m(A))\right]$$

It is also obvious that points B, C, D and E are located on shadow planes, and that the points A and F belong to the boundary of the participating medium. Of course point F and P can merge for object contained in the medium, and if it covers the entire scene, points A and O will also merge.

Finally, when considering a bounded medium, the equations are slightly different. The coefficient *r* and *x* in the exponentials of equation (1) must be the distance between the point *X* and the border of the medium boundary. In our method, we approximate *r* to the average distance *R* between a point located in the boundary and a point in the medium. So *R* is constant along the ray. The new value *xn* of *x* is computed on the fly and is also a constant along the ray. In this case, $L_m(P)$ becomes :

$$L_m(P) = \frac{k_t e^{-k_t(r+R-xn)}\Omega}{4\pi h}\int_{\theta_0}^{\theta_d} I_s(\theta+\beta)e^{-k_t h\frac{\sin(\theta)+1}{\cos(\theta)}} p(\theta+\frac{\pi}{2})d\theta$$

what only involves a change of coefficients *c*.

5. RENDERING ALGORITHM

5.1 Scenes Recovered by a Participating Medium

In this case, every view ray is contained entirely in the participating medium. The method is easy to implement and works as follows :

1. The silhouettes of every moving shadow caster are computed. If light is moving, every silhouette needs to be recomputed.
2. Scene is rendered using the conventional polygonal rendering method. Surface shadows can be obtained using shadow planes algorithms [He01, EK02]. The stencil buffer now contains lit areas of the scene. An ambient fog is added to take into account both absorption and multiple scattering.
3. Scene is rendered once more and medium contribution is computed for each vertex of the scene. Depth test is set to the equality. Only lit parts of the scene are rendered thanks to the stencil buffer.
4. Shadow planes, determined by the object's silhouettes, are sorted in a back to front order.
5. Shadow planes are rendered in that precise order. The depth test function accepts only planes that are closer to the camera. Front facing planes add their contribution when back facing planes subtract them. Stencil function is set to allow fragments if the stencil is equal to 1 for front facing planes and 0 for back facing ones. Front facing planes always increment the stencil buffer and back facing ones always decrement it.

All stages have to be done for each light source. Each stage is detailed in the following sections.

5.1.1 Computation of silhouettes

In our algorithm, we select some objects to be shadow caster. Their silhouettes are easily computed in determining all edges of their mesh common to a

 ISBN 1213-6972 ISBN 80-86943-09-7

front-facing triangle regarding the light position and one back facing it. Then all these edges are linked together if possible, and stored in a loop list. To obtain correct silhouettes, we need closed triangular meshes (2-manifold) for which connectivity informations are available. These conditions for the shadow casters are the ones indicated in [EK02].

Shadow planes are infinite quads formed by a silhouette edge and the light position. They are constituted by the two edge's vertices and two other points, projection of the previous vertices to infinity toward direction : light position - vertex (cf. [He91]). They are oriented toward the unshadowed area of the scene. As we need to compute the medium contribution on all shadow planes, it is necessary to use shadow plane silhouettes rather than the shadow planes of all little triangles. Of course, if the light does not move, only moving shadow caster silhouettes have to be computed. Finally, in case the input geometry is modified by graphics hardware, using displacement mapping for example, a solution to obtain silhouettes of all objects quickly and accurately can be found in [BS03].

5.1.2 Rendering the scene

The scene is rendered normally except for the light attenuation due to absorption and scattering induced by the participating medium. It multiplies the phong model used in the standard graphic pipeline a coefficient $e^{-k_t r}$ where kt is the extinction coefficient and r the distance from the lit point and the point light source. A simple vertex program can render this equation which differs from the traditional one only in the exponential attenuation.

In this stage we also add a fog effect to take into account both absorption and multiple scattering. We also compute the hard shadows and use the stencil algorithm and its improvements [He91, BS03] to do so. Indeed, they fit perfectly with our application since we already have the silhouettes. In the end of this stage, the stencil buffer contains the lit areas of the image. Until the end of the image rendering, the lighting is disabled.

5.1.3 Medium contribution of the scene

Still using stencil test, the scene is rendered once more to add, with additive blending, the medium contribution of every surface. This is simply done by computing equation (2) for each vertex. The depth test is set to the equality.

5.1.4 Sorting the shadow planes

Before rendering all shadow planes, we have to make sure that we will not render shadow planes, or part of them, that are themselves in shadow. If we do not care about this problem, it will create artifacts we call shadow in shadows, illustrated in figure 4. In the left image, we can see that the shadow of the top plane is propagated in the shadow of the bottom plane.

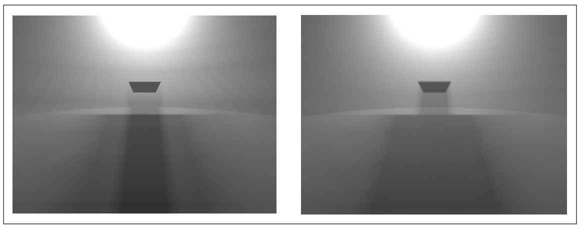


Figure 4: left : example of the shadows in shadows artifact. Right : a correct rendering

To prevent these artifacts we render the shadow planes, back- or front-facing, in a "back to front"' order and use the stencil buffer to avoid the rendering of shadowed shadow planes. The distance we defined for the back to front order depends on both camera and light positions. In two dimension, we can see in figure 5 that the plan (a line in 2D) created by the edge A (a point in 2D) must be rendered before the one created by B. And this one must be rendered before the shadow plane of edge C. This is true whatever the distance between the edge and the camera or between the edge and the light position. A simple realization of such a distance is to compute, for an edge P, the cosine between vectors $\overrightarrow{SO}$ and $\overrightarrow{SP}$ where O is the camera center, S the light position, and P a point belonging to the silhouette.

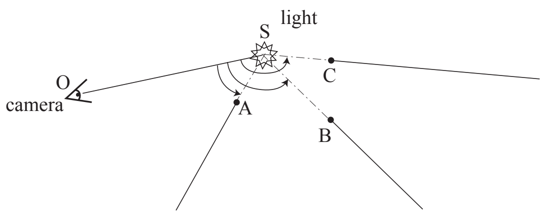


Figure 5: Ordering of shadow planes (in 2D)

We use the same ordering in 3D. In this case, the silhouette edges are segments. Since silhouettes are accurately meshed, these segments can be considered as points (only for ordering). Therefore, we compute the same cosine using as point P the center of the silhouette edge.

5.1.5 Rendering the shadow planes

We always keep the stencil we have obtained in the stage 2. Shadow planes are rendered in the order defined in the previous stage with the depth test function admitting only fragments that are closer to the camera.

The color attributed to the shadow planes – i.e. their contribution – are computed with exactly the same expression than for lit point of the scene in stage 3, i.e. using equation (2) for homogeneous point light. Front facing planes add their contribution and back facing planes subtract them.

We have to mesh the shadow planes to obtain accurate values of the medium contribution. They will be computed in each vertex of the mesh and the GPU will make the interpolation between them. According to the radial distribution of a point light, it is wise to mesh the shadow planes finely when close to the light and coarsely when far away. It is not necessary to subdivide the silhouette edge which has to be small.

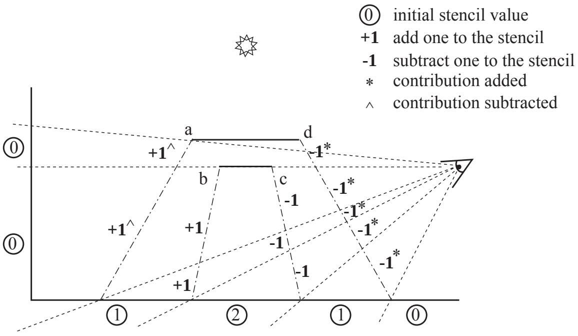


Figure 6: Use of the stencil buffer in the rendering of shadow planes

To take into account correctly the shadow in shadow problem, we use the stencil intensively. Front facing planes pass the stencil test if its value is one (representing shadowed area), and back facing ones passes if it equals 0 (value representing lit area). Ideally the back (resp. front) facing quads should always add (resp. subtract) one to the stencil buffer if it passes depth test. But unfortunately it is not possible to specify two different stencil functions when a fragment fails the stencil test depending of the result of the depth test. It imposes us to render the simple quad of the shadow plane with the stencil function set to always. Such problem will disappear when programmability of graphics card will involve the stencil test. Nevertheless this strategy works in all the case as illustrated in figure 6. The strategy indicated works if the camera is in the light. A slightly different strategy can be used when the camera is in shadow but the philosophy remains the same.

5.2 Rendering Several Bounded Participating Media

Several modifications have to be made to the previous algorithm to take into account boundaries of participating media and to avoid the rendering of each object and each shadow planes for every medium. Indeed, when several participating media exist, stages 3 to 5 need to be computed for each one of them. For simplicity we consider only convex participating media, and that we have a mesh representation of it.

First of all we will compute bounding boxes for each object and each participating medium. This is to avoid the rendering of objects that do not lay in the area of a participating medium in stage 3. We also check for each shadow plane if it is able to cut the participating medium.

In stage 3, we use the equivalent of a shadow volume algorithm to determine shadowed and lit area of the boundary of the participating medium. Then we render the lit areas of objects and of the medium boundary. Objects are rendered only if they belong to the medium bounding box. The front facing triangles of the medium boundary are rendered using the expression seen in section 4.

In stage 4, back-facing triangles of the medium boundary are also sorted and integrated in the order list. Since we use their center for the reference point P in the ordering, these triangles must be small. For each shadow plane, we determine if it is able to cut the medium bounding box. If not, it is removed from the sorted list to avoid unnecessary computation.

Finally the stage 5 remains the same, except that when a back facing triangle of the medium boundary is rendered, we set the stencil to 255 to avoid any further rendering in this area.

6. RESULTS

The previous algorithm has been implemented on a standard computer using a 2.6 GHz processor and an ATI 9800 graphic card. All images that we will present have a 800x600 resolution. We first compare our method with the work of Dobashi et al. [DY02] using their simple scene, a sphere beyond a spot light (cf. figure 7). In our case, the spot light is obtained by adding a cone above our point light. The silhouette has 32 edges which involves 32 shadow planes. Our rendering time is about 120 frames per second at resolution 800x600. For our algorithm, resolution is not really a problem. For example, the same scene using a 1024x768 resolution is rendered at 107 FPS. For the same test scene, Dobashi's algorithm achieves 12.5 FPS for a 450x300 resolution. This is mainly due to the accumulation of texture rendering inducing a high fill rate.

A drawback exists in our method which is only due to the clamping of the framebuffer. Indeed, when we render the contribution of the medium, it is possible that the final value added to the one present in the framebuffer exceeds 1. In that case, the value is clamped to 1 and if we subtract a medium contribution after that, the final result will be darker than it should be. However, this problem can be avoided in choosing reasonable intensity for the light

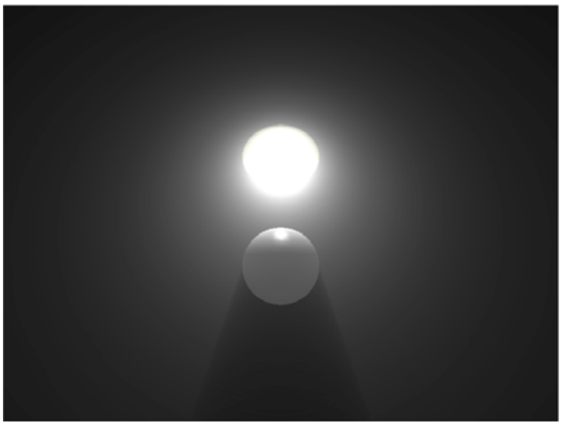


Figure 7 : Replicate of the Dobashi's scene

source, or in the future, using a float texture. Unfortunately, we do not have develop this yet.

We also present in table 1 the ratio of work loads for each stage (see 5.1). As expected, we can see that the computations of the shadow plane contributions represent the main cost of the whole process.

Stage	1	2	3	4	5
Fig 8 left	1%	8.7%	21.7%	9.6%	59%
Fig 8 right	1.7%	14%	33.6%	16%	34.7%

Table 1. Work loads for each stage.

We also present some snapshots of our animations. The first image in figure 8.a. is a simple scene, where two pens are bumping in front of a light. It illustrates a classical situation where well design 3D objects are moving and casting shadow. This scene is rendered at more than 35 fps. The image in figure 8.b. represents a simple scene with a box contained into three different participating media, one red, one blue and one green, moving before the light. Here we can clearly see the volumetric shadows of each participating media and how they blend together. As we use exact shadow planes no aliasing occurs. This case illustrates the ability of our algorithm to handle all the position between shadow planes and the boundary of a participating medium. Figure 8.c. is a snapshot from a animation where the light is moving, and its color is also changing. We have chosen this scene because it contains a lot of shadow planes. Finally figure 1.c., in the first page of this paper, is also a snapshot to illustrate the use of our algorithm when light is moving in a complex scene, containing around 100 000 triangles. Table 2 presents the FPS and the number of triangles of those scenes.

Scene	Fig 8a	Fig 8b	Fig 8c	Fig 1
FPS	23	35	25	12
Nb. triangles	34 549	14 785	20 747	107 514

Table 2. FPS and number of triangles of scenes.

7. CONCLUSION

We have presented a new real time algorithm that is able to compute the single scattering of one or several participating media. Our algorithm is fast enough to handle more than 25 frames per second for moderately complex scenes, which is an improvement over other atmospheric scattering algorithms, especially when a medium covers the whole scene. As outlined above, the only computations that we have done in software are the participating medium contributions and the ordering and the computation of shadow planes. Moreover, we plan to design vertex and fragment shaders to make the graphic card computes the participating medium contributions. We also want to point out that our algorithm does not create any sampling aliasing artifact, for both surface and volumetric shadows, thanks to the use of exact shadow planes.

As shadow planes have become more popular recently, we think that our algorithm fit perfectly with this kind of approach and is well adapted to the growing capacities of graphics hardware. For example, the final improvement of the algorithm would be to compute soft surface shadows and soft volumetric shadows. For this goal we can take inspiration of the algorithm [AM03]. Finally, both clustering and culling approaches will greatly speed up this already fast algorithm.

8. REFERENCES

[AM03] Assarson U., Möller T.A., A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware, In proceedings of SIGGRAPH’03, Computer Graphics, vol. 22 (3), pp. 511-520

[BR98] Behrens U., Ratering R. , Adding Shadows to a Texture-based Volume Renderer. In proceedings of 1998 symposium on Volume Vizualisation , 1998, pp. 39-46

[BS03] Brabec S., Seidel H.P., Shadow Volumes on Programmable Graphics Hardware. In proceedings of Eurographics’03, 2003, vol. 22(3)

[DK00] Dobashi Y., Kaneda K., Yamashita H., Okita T., Nishita T., A Simple, Efficient Method for Realistic Animation of Clouds. In proceedings of SIGGRAPH’00, Computer Graphics, 2000, pp. 19-28

[DY00] Dobashi Y., Yamamoto T., Nishita T., Interactive Rendering Method for Displaying Shafts of Light. In proceedings of Pacific Graphics 2000, pp. 31-37.

[DY02] Dobashi Y., Yamamoto T., Nishita T., Interactive Rendering Method of Atmospheric Scattering Effects Using Graphics Hardware. In proceedings of Graphics Hardware 2002, 2002, pp. 99-107.

[EK03] Everitt C., Kilgard M., Practical and Robust Shadow Volumes, Nvidia white paper, 2003 http://developer.nvidia.com/object/robust_shadow_volumes.html.

[Ev02] Everitt C., A Fast Algorithm for Area Light Source Using Backprojection. In proceedings of SIGGRAPH’94, Computer Graphics, 1994, pp. 223-230

[FM97] Foster N., Metaxas D., Modeling the Motion of a Hot, Turbulent Gas. In proceedings of SIGGRAPH’97, Computer Graphics, 1997, pp. 181-188

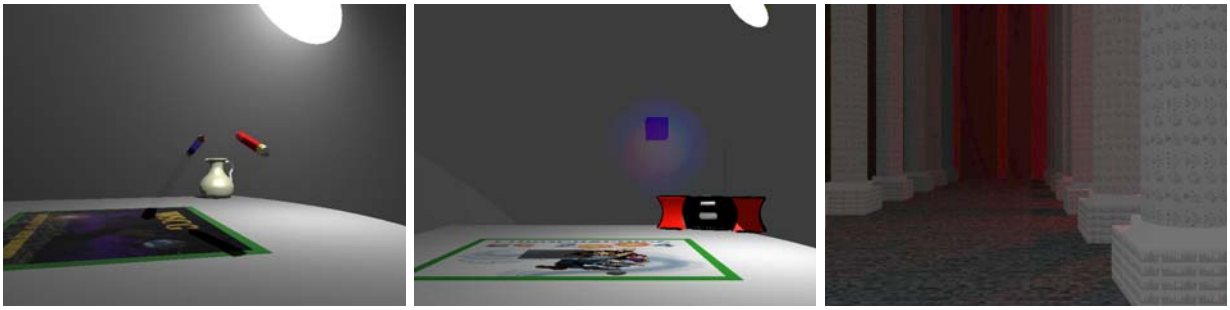


Figure 8. a. (left) Two pen moving. b. (center) Three participating media moving. c. (right) Light is moving in a relatively complex scene.

[FS01] Fedwik R., Stam J., Jensen H., Visual Simulation of Smoke. In proceedings of SIGGRAPH'01, Computer Graphics, 2001, pp. 15-22.

[He91] Heidman T., Real Shadows Real Time. In IRIS Universe (1991), vol. 18, pp 28-31

[HP02] Hoffman N., Preetham A., Rendering Outdoor Light Scattering in Real Time. ATI white paper, 2002. www.ati.com/developer/dx9/ATI-LightScattering.pdf

[JC98] Jensen H., Christensen P., Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps. In proceedings of SIGGRAPH'98, Computer Graphics, pp 311-320

[Le01] Lecocq P., Simulation d'éclairage temps réel par des sources lumineuses mobiles et statiques : outils pour la simulation de conduite. PhD Thesis of the University of Marne-la-Vallée, 2001. Samples of this thesis can be found here : http://igm.univ-mlv.fr/~biri/

[LG02] Lefebvre S., Guy S., Volumetric Lighting and Shadowing, NV30 Shader, 2002. lefebvre.sylvain.free.fr/cgshaders/vshd/vshd.html

[LM00] Lecocq P, Michelin S., Arquès D., Kemeny A., Mathematical Approximation for Real Time Rendering of Participating Media considering the luminous intensity distribution of light sources. In proceedings of Pacific Graphics 2000. pp 400-401.

[LW96] Lafortune E., Willems Y., Rendering Participating Media with Bidirectional Ray Tracing. In proceedings of 6th Eurographics Workshop on Rendering, june 1996, pp. 92-101.

[Ma94] Max N., Efficient Light Propagation for Multiple Anisotropic Volume Scattering. In proceedings of 5th Eurographics Workshop on Rendering, 1994, pp. 87-104

[Me01] Mech R., Hardware-Accelerated Real Time Rendering of Gaseous Phenomena. In Journal of Graphics Tool, 2001, vol. 6(3), pp. 1-16

[ND96] Nishita N., Dobashi Y, Nakamae E., Display of Clouds Taking into Account Multiple Anistropic Scattering and Skylight. In proceedings of SIGGRAPH'96, june 1996, pp. 379-386

[NM87] Nishita N., Miyawaki Y, Nakamae E., A shading model for atmospheric scattering considering luminous distribution of light sources. In proceedings of SIGGRAPH'97, Computer Graphics, vol. 21(4), pp. 303-310

[NM01] Nulkar M., Mueller K., Splatting with shadows. In proceedings of Volume Graphics 2001, pp. 35-49

[On05] S. O'Neil, Accurate atmospheric scattering, In GPU Gems 2, Addison Wesley, march 2005, pp. 253-268.

[PM93] Pattanaik S., Mudur S., Computation of global illumination in a participating medium by monte carlo simulation. In The journal of Visual and Computer Animation, 1993, vol 4(3), pp. 133-153

[RT87] Rushmeier H., Torrance K., The zonal method for calculating light intensities in the presence of participating medium. In proceedings of SIGGRAPH'87, computer graphics vol 21(4), pp. 293-302.

[Ru94] Rushmeier H., Rendering participating media : problems and solutions from application areas. In proceedings of 5th Eurographics Workshop on Rendering, june 1994, pp. 35-56.

[SH92] Siegel R., Howell J., Thermal Radiation Heat Transfer. 3rd ed. Hemisphere Publishing, 1992.

[SR05] Sun B., Ramamoorthi R., Narasimhan S.G., Nayar S.K., A practical analytic single scattering model for real time rendering. In proceedings of SIGGRAPH'05, Computer Graphics; 2005, vol 24(3), pp. 1040-1049.

[St99] Stam J., Stable fluids. In proceedings of SIGGRAPH'99, Computer Graphics, 1999, pp. 121-128.

[WE98] Westermann R., Ertl T., Efficiently using graphics hardware in volume rendering applications. In proceedings of SIGGRAPH'98, Computer Graphics, 1998, pp. 169-177

9. ANNEXES

Expression of coefficients c for classical phase functions. Isotropic phase function :

$$c_0 = e^{-k_t h}$$
$$c_1 = -k_t h \; e^{-k_t h}$$
$$c_2 = \left(k_t^2 h^2 - k_t h\right) e^{-k_t h}$$
$$c_3 = \left(-\frac{(k_t h)^3}{6} + \frac{k_t^2 h^2}{2} - k_t h\right) e^{-k_t h}$$
$$c_4 = \left(\frac{(k_t h)^4}{24} - \frac{(k_t h)^3}{4} + \frac{11\, k_t^2 h^2}{24} - \frac{5 k_t h}{24}\right) e^{-k_t h} \; \ldots$$

Rayleigh phase function :

$$c_0 = \frac{3}{4}\, e^{-k_t h}$$
$$c_1 = \left(-\frac{3\; k_t h}{4}\right) e^{-k_t h}$$
$$c_2 = \left(\frac{3\; k_t^2 h^2}{8} + \frac{3}{4}\right) e^{-k_t h} \; \ldots$$

Hazzy phase function :

$$c_0 = \frac{265}{256}\, e^{-k_t h}$$
$$c_1 = \left(-\frac{265 k_t h}{256} + \frac{9}{36}\right) e^{-k_t h}$$
$$c_2 = \left(\frac{265 k_t^2 h^2}{256} - \frac{121\; k_t h}{512} + \frac{63}{64}\right) e^{-k_t h} \; \ldots$$

Murky phase function :

$$c_0 = \frac{2147483673}{2147483648}\, e^{-k_t h}$$
$$c_1 = \left(\frac{2147483673\; k_t h}{2147483648} - \frac{25}{367108864}\right) e^{-k_t h}$$
$$c_2 = \left(\frac{2147483673\; k_t^2 h^2}{4294967296} - \frac{2147482073\; k_t h}{4294967296} + \frac{775}{134217728}\right) e^{-k_t h} \; \ldots$$

Visual Exploration of Seismic Volume Datasets

Timo Ropinski
Institut für Informatik
WWU Münster
Einsteinstraße 62
48149 Münster, Germany
tr@math.uni-muenster.de

Frank Steinicke
Institut für Informatik
WWU Münster
Einsteinstraße 62
48149 Münster, Germany
fsteini@uni-muenster.de

Klaus Hinrichs
Institut für Informatik
WWU Münster
Einsteinstraße 62
48149 Münster, Germany
khh@uni-muenster.de

ABSTRACT

This paper introduces a novel method supporting the interactive exploration of volumetric subsurface data. To facilitate better insights into the datasets we propose the application of focus and context visualization metaphors. Using these metaphors users can emphasize arbitrary parts of a dataset or remove occluding information interactively to focus on the region of interest. In addition to these visualization issues we will explain how the focus and context metaphors can be combined with VR-based interaction techniques to allow the efficient exploration within more immersive VR environments. In particular, we will discuss how to control the focus and context metaphor to highlight the region of interest in combination with the usage of visual bookmarks to track potentially interesting parts within large volumetric subsurface datasets.

Keywords
Subsurface Exploration, Seismic Volume Data, Focus and Context Visualization, VR.

1 INTRODUCTION

With decreasing availability of fossil fuels such as oil and gas, the demand for efficient techniques to locate the remaining reservoirs is rising. Drilling for these resources is a very cost intensive process, e.g., according to the Joint Association Survey on Drilling Costs drilling one well can cost up to several million US dollars. Hence, the oil and gas industry tries to reduce drilling costs, by using extensive computer-aided analysis and exploration to make better predictions regarding the location of subsurface reservoirs. Also more precise predictions of natural disasters such as earthquakes can be obtained by computer aided subsurface analysis. The most common approach to perform this subsurface analysis is interactive 3D exploration of subsurface structures on the basis of seismic datasets.

Seismic datasets are acquired indirectly. Detonating explosives emit sound waves, which are propagated through the subsurface. The reflections of those sound waves are measured with special receivers located at the surface. Thus geological layers, which reflect the sound waves, can be identified based on the time elapsed before a signal is received as well as the signals attenuation. The resulting information is encoded in a 3D volume composed of discrete samples, each representing the amplitude reflected beneath the surface. This usually very large 3D volume is often stored as a SEG-Y dataset [SEG02], and the acquired amplitudes can be visualized using different transfer functions to represent the geological structure.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

Although the idea of 3D GIS has been proposed [KLR$^+$03], most current 3D GISs adapt the concepts of polygonal representations for visualizing geographic information and are therefore not sufficient for analyzing volumetric seismic datasets. For these reasons GISs supporting volumetric methods have been developed, which are highly specialized applications designed for the purpose of exploring seismic datasets [Geo06, VGe06, Fle06]. In conjunction with these systems, virtual reality (VR) systems support exploration of seismic datasets. VR systems provide three major advantages compared to desktop-based visualization systems. By using stereoscopic projection techniques, spatial perception is improved and thus aids visual comprehension. Furthermore, by using appropriate tracking technologies, the user can perform intuitive 3D interactions, whereas desktop-based systems usually support only 2D input devices.

To benefit from these capabilities provided by VR systems, the developer of a visualization application has to consider VR design issues, e.g., the interactions which can be performed have to be adapted to VR-based interactions. To grant an efficient exploration of seismic datasets it is important to obtain interactive frame rates within the application in order to give immediate visual feedback when the user has performed an interaction. In the context of this paper we faced three major challenges all leading to a reduced frame rate: (1) seismic datasets are usually very large; (2) the high screen resolutions of current displays; (3) for stereoscopic viewing in VR systems the scene has to be rendered twice, i.e., once for each eye. Therefore it is important to use state-of-the-art volume rendering techniques which exploit the features provided by current graphics hardware to achieve interactive frame rates.

In this paper we propose a novel method for VR-based exploration of seismic datasets. Our approach supports efficient exploration of seismic volume datasets by using focus and context visualization metaphors in combination with intuitive and efficient VR-based interactions. By using these interaction concepts the user can emphasize the region of interest inside the dataset interactively. This region of interest is specified by a *lens volume*, which has an arbitrary convex 3D shape. Inside the region of interest a different visual representation is used for visualization and thus aids comprehension of the dataset without loosing the contextual information outside the region of interest (see Figure 1). By using specialized 3D interaction metaphors the user can switch between different lens shapes as well as visual representations. Furthermore, it is possible to set *spatial visual bookmarks* at certain points of interest, which can be accessed later on using a guided VR-based exploration-metaphor. For visualizing the datasets we use GPU-based ray-casting [RGWE03, KW03] which enables interactive frame rates while rendering the scene stereoscopically and with a high resolution. To use different visual representations within the region of interest we have extended the GPU-based ray-casting technique to support different rendering modes when visualizing a volume dataset.

This paper is structured as follows. In the next section we discuss related work while Section 3 briefly outlines our extension of GPU-based ray-casting and the kinds of visual representations we have found helpful for exploring seismic datasets within our prototype system. In Section 4 the VR-based interaction techniques used to alter the region of interest are explained in detail. The paper concludes in Section 5 by giving a short overview of the presented concepts.

2 RELATED WORK

Visualization techniques for representing volumetric phenomena have advanced in the past years. While most of the proposed techniques have been developed for medical applications, Ma and Rokne [MR04] present a good overview of visualization techniques used to visualize seismic volume datasets. Furthermore, various software applications, e.g., Fledermaus[1], VoxelGeo[2], and GeoProbe[3], that use seismic data to compile and process subsurface visualizations are available. With these applications, geo-scientists are able to process and visualize seismic data on commodity hardware. However, even with these highly evolved applications and the incorporated visualization techniques the information contained in volumetric data can be overwhelming during exploration.

In medical visualization a lot of work has been done to support the generation of focus and context visualization [HMBG01, VKG05, BGKG05]. In 2005 two software systems have been presented, which allow interactive generation of illustrative visualizations of medical volume data [BG05, SES05]. Both systems incorporate recent focus and context visualization techniques. A general approach to support focus and context visualization by applying the magic lens metaphor to volume datasets can be found in [WZMK05]. Furthermore, Diepstraten et al. [DWE03] have proposed a set of guidelines for generating cutaway illustrations, which can be used to provide the observer insights to an otherwise opaque object.

Immersive visualization and interaction technologies, allowing multidisciplinary teams to explore the subsurface data in a more intuitive and efficient manner, help to increase the value of such enormous amounts of volumetric datasets. A wide range of visualization systems and input devices, which have originally been developed for applications outside of the geo-scientific industry, have been adapted to support users when exploring seismic datasets. Among others, these systems include immersive projection-based VR system environments such as the CAVE, curved and flat-wall displays, desk- or workbench-type systems as well as head-mounted devices ([HBP+02],[CNSD+92]). The best choice of a certain visualization system depends on the task to be accomplished within the system. However, especially the semi-immersive responsive workbench ([KBF+95]) has proven its potential as table-top metaphor for the exploration of geo-spatial data ([FBZ+99], [KHJ00]). The capability to mount the workbench in a horizontal position allows to visualize geo-spatial data in an intuitive way. Although the visualization of seismic datasets within immersive VR systems enables an advanced exploration of the datasets and improves the information retrieval, interaction with the data within current systems is usually limited to mouse, keyboard and/or joystick-type devices, and often requires an experienced navigation expert to pilot other viewers through the visualization. However, since semi-immersive and immersive VR systems allow to explore and process a huge amount of data, recent research approaches review innovative interaction technologies to enable an intuitive exploration and manipulation of geo-spatial data ([MEH+99]).

Fröhlich et al. ([FBZ+99]) have shown, how the exploration of seismic datasets can benefit from the use of volumetric lenses. However, their approach supports

[1] http://www.ivs3d.com/products/fledermaus/
[2] http://www.paradigmgeo.com/products/voxelgeo.php
[3] http://www.magic-earth.com/geo.asp

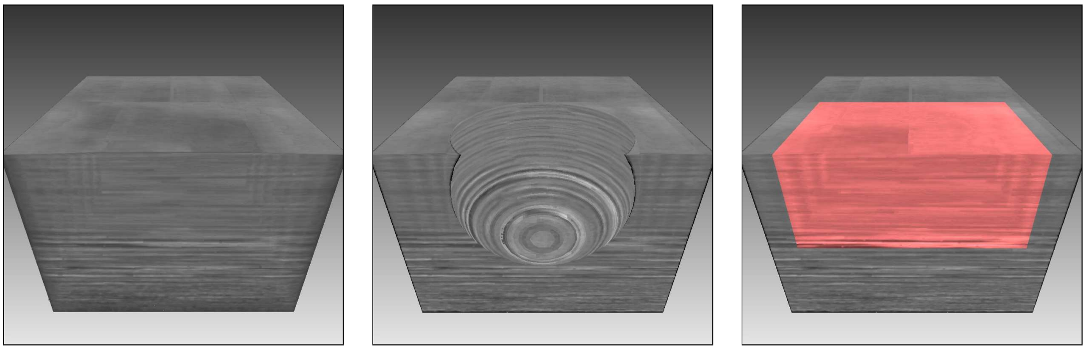


Figure 1: Application of focus and context visualization metaphors to a seismic volume dataset. Seismic dataset rendered without using a lens (left), with a spherical clipping lens applied (middle) and with a cuboid lens altering the used transfer function (right).

only lenses having cuboid geometries. Because the surfaces of a cube are given by orthogonal planes, the visualization is similar to the common slicing approach where multiple clipping planes are used to reveal inside structures ([CGH86]). Therefore the technique introduced by Fröhlich et al. can be considered as a generalization of this slicing approach.

However, due to the huge amount of volumetric datasets tracking of the regions of interest is often difficult. Once found regions may get lost since no VR techniques support the geo-scientist when accessing such already explored regions. Concepts for storing regions as well as positions in 3D environments have previously been introduced for guided navigation tasks ([SGLM03], [Döl05]). These approaches allow the user to store certain positions and to define camera paths leading through these so called *bookmarks* placed in the virtual environment. For volume exploration no comparable approaches exist.

3 VISUALIZATION OF VOLUMETRIC SUBSURFACE DATA

In this section we shortly summarize our extension to the GPU-based ray-casting technique [RGWE03] to support different visual representations of the region of interest within a volume dataset. A more detailed explanation of our extension can be found in [RSH05].

GPU-based ray-casting supports very efficient volume rendering on commodity graphics hardware by casting rays through the volume dataset which is represented by a 3D texture.

To apply a different visual representation for the region of interest inside the lens volume, the voxels contributing to a pixel in image space need to be distinguished whether they are inside or outside the lens volume. We determine this voxel classification before rendering the dataset, since it results in better performance, because less per-fragment operations are needed during rendering compared to an alternative approach where the regions are distinguished during rendering. Therefore, in our approach a ray cast through a volume dataset which is intersected by a convex lens volume is split into at most three different sections, i.e., one section in front of the lens, one inside the lens and one behind the lens (see Figure 2). This leads to the three view-dependent regions *region*$_0$, *region*$_1$ and *region*$_2$.

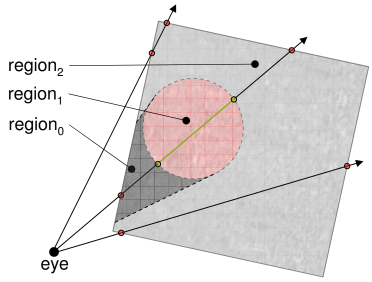


Figure 2: Scheme showing the three regions defined by the lens volume.

Our algorithm renders these view-dependent regions in three subsequent rendering passes starting with *region*$_0$. Because the view-dependent regions of the volume dataset are determined before accessing the dataset itself, rendering can be performed very fast, since only the usually simple shaped lens geometry and the bounding box of the volume dataset are considered during this computation. In the next subsections we are going to discuss concrete lenses which are useful for 3D exploration of seismic datasets.

3.1 Occlusion Lens

One problem occurring in the visualization of information associated with volumetric data is that usually no insight view can be provided. Especially when navigating through a dense volume dataset the view of the camera will always be occluded. To avoid this problem we propose an *occlusion lens* which renders those parts of the volume dataset transparently that occlude the region of interest (see Figure 3). Therefore when rendering *region*$_0$ we set the degree of transparency proportional to the amount of volume data occluding the region of interest, i.e., the number of voxels encountered by the ray. This effect does not result in any

 ISBN 1213-6972 ISBN 80-86943-09-7

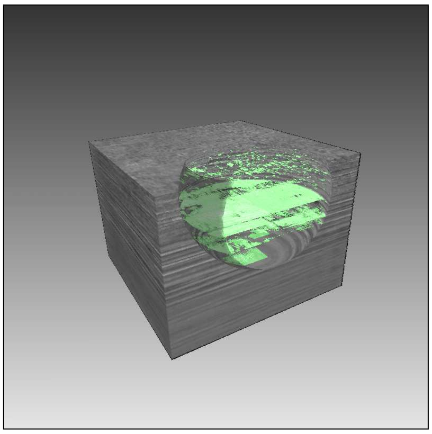


Figure 3: Application of an occlusion lens which renders the parts occluding the region of interest transparently. Inside the lens subsurface structures having a certain intensity are highlighted.

performance penalty compared to direct volume rendering, because during ray traversal instead of setting the voxel's alpha value based on the corresponding scalar value simply the number of samples already processed is used to determine the alpha value.

Another lens type, which we also classify as occlusion lenses is the clipping lens, since it assists in exploring the information on the surface of the lens volume which would be occluded otherwise. Such a lens is shown in Figure 1 (middle) where a spherical clipping lens is used to reveal information hidden inside the dataset. For an easier identification of the clipping region its silhouette edges are enhanced.

3.2 Slicing Lens

A *slicing lens* incorporates clipping-based rendering of slices as visual representation (see Figure 4 (right)). When applying slice rendering surfaces sliced through the volume dataset are displayed. In our prototype application the number of slices and the distance between the slices can be chosen interactively and the visualization provides immediate visual feedback. Thus, it is possible to visualize certain structures based on this rendering technique. As shown in Figure 4 (left) when applying slice rendering to the entire dataset it is difficult to comprehend spatial relationships. Because parts of the dataset are invisible to the user it is difficult to visually comprehend the structure of the dataset and thus the location of the slices. In Figure 4 (right) this spatial relationships can be detected more easily because contextual information is maintained outside the lens. Furthermore the cross-section displayed on the surface of the lens gives an important cue to localize the visualized slices in relation to the rest of the dataset.

3.3 Object Emphasizing Lens

Another lens which improves interactive exploration of seismic volume datasets allows to emphasize objects of interest by applying image-based silhouette highlighting in combination with a different transfer function. Although in most volume visualization applications the same transfer function is used for the entire dataset, it may be desirable to use different transfer functions in different parts of the dataset, e.g. to identify special regions of interest [SML$^+$99]. Thus, it is possible to reveal structures within the data without changing the context, since in the parts outside the lens volume the original transfer function is retained. In Figure 5 (right) an example is shown where the parts of the dataset lying outside the lens are rendered using direct volume visualization techniques, whereas the parts inside the lens are displayed using a different threshold value as well as a different transfer function to highlight certain objects. In the shown example we use a 1D transfer function inside and outside the focus region, but it would also be possible to combine 1D and 2D transfer functions. Furthermore, the color used for edge-enhancement, the threshold value as well as the transfer function used inside the lens volume can be exchanged interactively. Hence, geo-scientists can either start with a preset threshold and explore the highlighted structures, or they can alter the threshold interactively to find an adequate value for identifying certain structures. By combining the edge-enhancement technique with the altered transfer function objects of interest and their structure can be identified more easily. It can be seen in Figure 5 (left) that using the technique for the entire dataset results in more difficult spatial comprehension.

3.4 Level of Detail Lens

In addition to applying the visual representations as described above, the introduced algorithm allows to further enhance rendering performance. This may be necessary when dealing with datasets having dimensions greater than the maximum dimensions allowed for 3D textures on certain graphics hardware. To enhance performance, a different level of detail (LoD) is used inside the region of interest compared to the LoD outside the region. This is achieved by using a different sampling rate. Figure 6 shows the application of a lens with a different transfer function used for the visualization inside the lens. In Figure 6 (left) the same LoD is used inside and outside the lens, whereas in Figure 6 (right) the LoD outside the lens is coarsened by using half the sampling rate compared to the visualization inside the lens. Usually geo-scientists are interested in details within the region of interest; the parts outside the region of interest are not in focus and may therefore be rendered using a lower sampling rate. Furthermore the LoD outside the lens can be increased adaptively over time when rendering is idle.

In addition to the lenses described in this section several other lenses and even combinations of differ-

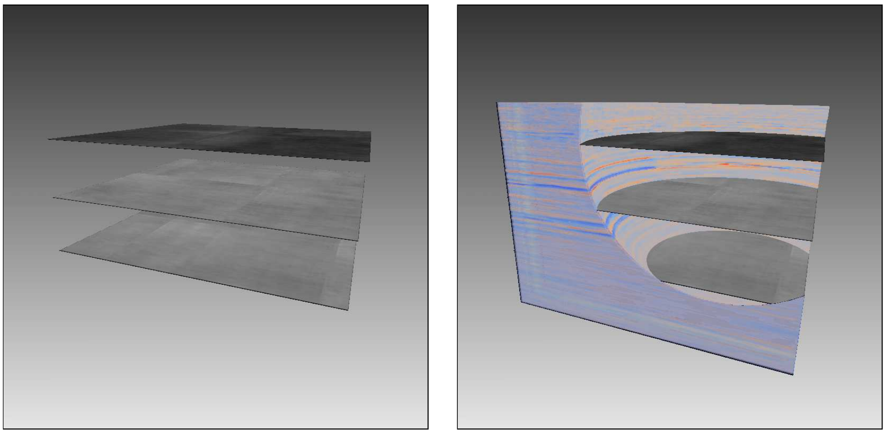


Figure 4: Slice rendering of a seismic volume dataset (left). Slice rendering only applied within the lens volume aids to comprehend spatial relationships (right).

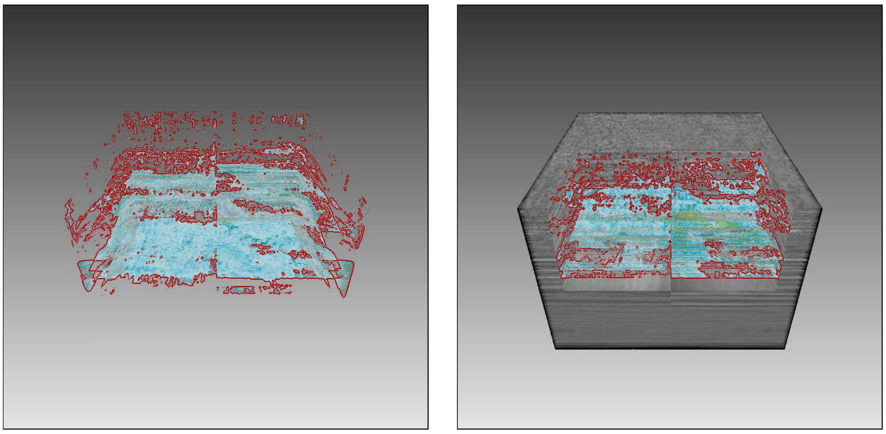


Figure 5: Seismic dataset rendered with image-based edge-enhancement and different transfer function (left) and with edge-enhancement and different transfer function only within the region of interest (right).

ent visual representations inside the lens are possible, e.g., isosurface rendering of translucent surfaces can be combined with regular volume rendering techniques. However, the lenses presented in this section provide a good understanding of the possibilities when applying focus-based visualization techniques to seismic volume datasets.

4 INTERACTION TECHNIQUES FOR VR-BASED EXPLORATION

Although VR technologies provide an excellent alternative to allow more immersive and advanced exploration of large datasets, when using seismic exploration software interaction is often restricted to standard desktop interaction concepts, i.e., interactions are performed via mouse or keyboard devices. This leads to a loss of immersion, since geo-scientists have to accomplish 3D interaction using 2D devices. In this section we propose a VR-based interaction technique that enables users to explore volumetric subsurface data in a very intuitive and efficient way.

4.1 Visualization in a Responsive Workbench Environment

Due to the horizontally mountable projection screen, the semi-immersive workbench has been proven to be advantageous for the exploration of geo-spatial data. We stereoscopicaly display subsurface data in our workbench environment by assigning a *negative parallax* such that the seismic dataset appears on top of the projection screen (see Figure 7). Thus, it is possible to inspect the dataset from all sides. If the height of the dataset exceeds the maximum height that can be visualized above the projection screen, parts of the dataset have to be displayed with *positive parallax*, i.e., below the projection plane.

To enable interaction with surface and subsurface datasets projected on the responsive workbench an optical tracking system determines the position and the orientation of the user's head and input devices. Tracking of the position and orientation of the input devices enables users to interact with the virtual scene. Manipulations can be performed by using *virtual hand metaphors* ([Min94]), i.e., users can directly pick and manipulate objects with the hand or they can move the described

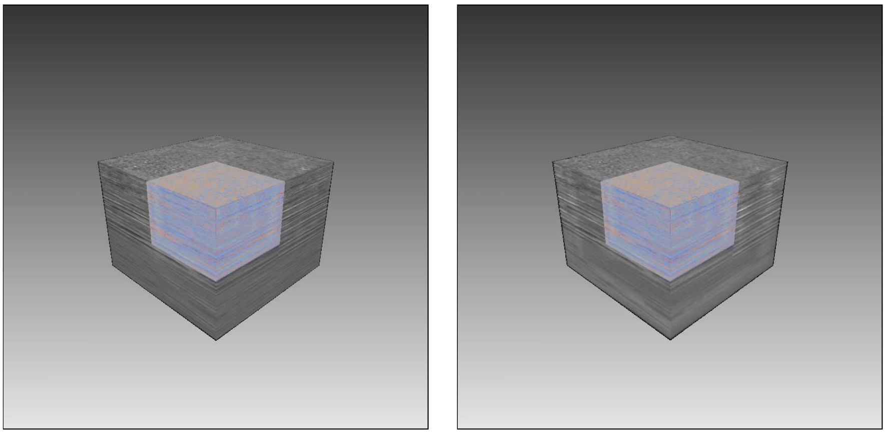


Figure 6: A lens applying a different transfer function. The same LoD is used inside and outside the lens (left). The LoD is coarsened outside the lens by using half the sampling rate (right).

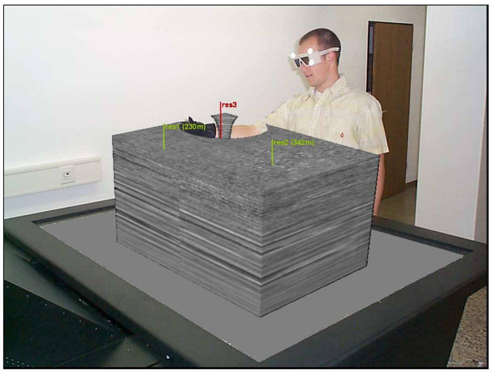


Figure 7: User controlling a spherical clipping lens applied to a seismic volume dataset in a responsive workbench environment. The visual bookmarks have been aligned to the image plane of the observer camera.

lens volume attached to an input device through the subsurface data.

Virtual hand metaphors are not sufficient to allow interaction with parts of a seismic dataset displayed below the projection screen, therefore different strategies must be applied in this context.

4.2 Controlling the Lens

The described lens metaphor offers a comfortable way to explore seismic datasets by visually intruding into the volume. Especially in combination with a stereoscopic projection the user experiences an immersive visualization of complex structures, which improves the cognition of spatial data. To support this immersion, we provide a very intuitive and efficient VR-based control of the lens. The position and orientation of a tracked input device, e.g., a user's hand, are used to move the lens accordingly. Thus, a very intuitive and natural mechanism for controlling the lens through the volumetric dataset is ensured. However, the usage of this interaction metaphor has its limitations. In regions where the seismic data are displayed below the projection screen, distant object interaction, i.e., interaction with data positioned outside the immediate reach of the user, must be guaranteed.

To enable moving the lens into the aforementioned regions, which are not accessible to the user, we provide the concept of changing the control/display (C/D) ratio. The C/D ratio controller adjusts the ratio determining the relationship between input device movements and the motion of the controlled virtual object, i.e., the lens volume. This adjustment is determined by the input device's position relative to the projection plane, i.e., the closer the hand is to the projection plane the higher this ratio is set. Thus, it is possible to make the movements of the virtual input device less sensitive compared to the user's hand movement or to scale the movements of the user's hand, e.g., for distance object manipulation [BJH01]. Hence in our setup a translation transformation is applied as offset between the real input device and the lens volume depending on the depth of the seismic volume dataset displayed below the projection screen. Thus, an intuitive and efficient distant control of the lens volume is possible.

However, when displaying subsurface data below the projection screen, a side view onto the seismic dataset is not possible due to physical constraints of the hardware setup. Therefore, we have implemented a mechanism to lift the seismic dataset out of the workbench frame to make the lower portions of the volume visible from the side. If the height of the dataset exceeds the maximum height that can be visualized above the projection screen this may result in truncation of the upper parts of the volume dataset.

4.3 Spatial Visual Bookmarking

Since, usually the amount of seismic data acquired for an arbitrary area is very large, localization of regions of interest can be a difficult task. Therefore, we provide a mechanism to support the user by guiding him during the exploration, i.e., the geo-scientist is assisted in locating regions of interest. Once such a region is identified, the geo-scientist can mark this region and make it accessible later on. Regions of interest can be stored by bookmarking them with a visual hint. In order to support an easy identification by the user textual information about the regions of interest can be attached to the bookmarks.

Individual bookmarks are markers positioned within the seismic dataset, which are easily accessible. When the lens is moved by the user through the dataset emphasizes a special region of interest, an individual bookmark can be defined, e.g., by pressing a button of the input device, and the user gets a visual hint about the marker's position. This approach ensures, the management of bookmarks, e.g., bookmarks can be created, deleted or repositioned.

4.4 Accessing the Bookmarks

Once the bookmarks are defined we use the so called *improved virtual pointer (IVP)* metaphor ([SRH05]) to select a bookmark to position the lens at the appropriate location. The IVP metaphor is a ray-casting-based selection and manipulation metaphor. With ray-casting techniques a selection can be performed when the ray extending from the user's input device hits a selectable object. When attempting to access a spatial visual bookmark geo-scientists benefit from the usage of the IVP metaphor since it provides a very efficient mechanism to aim at small distant objects, such as spatial visual bookmarks. In contrast to other pointer metaphors ([Min94]) selection of a certain object does not require an exact hit of that object with the ray. Thus, the user roughly points the input device to a visual bookmark, and the IVP metaphor determines the bookmark closest to the ray. When using the IVP metaphor distant objects are easy to select and access to bookmarks deep inside the seismic dataset is ensured.

In Figure 7 a user is exploiting the explained interaction concepts in our responsive workbench environment. The seismic dataset is projected onto the projection screen of the workbench, and the user defines and accesses several individual bookmarks within the dataset.

5 CONCLUSION AND FUTURE WORK

In this paper we have presented a novel method allowing interactive VR-based exploration of volumetric seismic datasets. We have proposed the usage of focus-based visualization techniques to ease the identification of certain structures within seismic dataset. Furthermore, we have introduced interaction techniques developed to control the used visualization metaphors intuitively within VR environments. By allowing geo-scientists to set spatial visual bookmarks the user is guided during exploration and can backup previously discovered results.

In the future we will define a taxonomy of lenses used in volumetric visualization applications to assist application developers to choose the optimal lens for their purposes. Furthermore, we are going to conduct a user study with a formal statistical analysis to determine usability as well as efficiency of the proposed concepts.

6 ACKNOWLEDGMENTS

We would like to thank the *Deutsche Montan Technologie* (DMT) as well as the *Deutsche Steinkohle AG* (DSK) for providing us seismic volume datasets and giving us insights into the computer-aided seismic exploration process. Furthermore, we are thankful for the valuable comments of the reviewers which helped to improve the paper.

REFERENCES

[BG05] S. Bruckner and E. Gröller. Volumeshop: An interactive system for direct volume illustration. In H. Rushmeier C. T. Silva, E. Gröller, editor, *Proceedings of IEEE Visualization 2005*, pages 671–678, October 2005.

[BGKG05] S. Bruckner, S. Grimm, A. Kanitsar, and E. Gröller. Illustrative context-preserving volume rendering. In *Proceedings of EuroVis 2005*, pages 69–76, May 2005.

[BJH01] D. Bowman, D. Johnson, and L. Hodges. Testbed evaluation of virtual environment interaction techniques. *Presence: Teleoperators and Virtual Environments*, 10(1):75–95, 2001.

[CGH86] M. P. Curtis, A. C. Gerhardstein, and R.E. Howard. Interpretation of large 3-D data volumes. In *Expanded Abstracts of the Society of Exploration Geophysicists*, pages 497–499, 1986.

[CNSD+92] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. Kenyon, and J. C. Hart. The CAVE, audio visual experience automatic virtual environment. *Communications of the ACM*, pages 64–72, June 1992.

[Döl05] J. Döllner. Constraints as means of controlling usage of geovirtual environments. *Cartography and Geographic Information Science*, 32(2):69–80, 2005.

[DWE03] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway illustrations. In *EUROGRAPHICS*, pages 523–532, 2003.

[FBZ+99] B. Fröhlich, S. Barrass, B. Zehner, J. Plate, and M. Göbel. Exploring geoscientific data in virtual environments. In *Proceedings of the Conference on Visualization (VIS99)*, pages 169–174. IEEE Press, 1999.

[Fle06] Fledermaus. Technical report, IVS, 2006.

[Geo06] GeoProbe. Technical report, Magic Earth, 2006.

[HBP+02] N. R. Hedley, M. Billinghurst, L. Postner, R. May, and H. Kato. Explorations in the use of augmented reality for geographic visualization. *Presence: Teleoperators and Virtual Environments*, 11(2):119–133, 2002.

[HMBG01] H. Hauser, L. Mroz, G.I. Bischi, and E. Gröller. Two-level volume rendering. *Transactions on Visualization and Computer Graphics*, 7:242 – 252, July 2001.

[KBF+95] W. Krüger, C. Bohn, B. Fröhlich, H. Schuth, W. Strauss, and G. Wesche. The responsive workbench: A virtual work environment. *IEEE Computer*, 28(8):42–48, 1995.

[KHJ00] F. Kuester, B. Hamann, and K. I. Joy. Interactive two-handed terrain and set design in immersive environments. *Proceedings of Tenth International Conference on Artificial Reality and Tele-Existence*, pages 31–35, 2000.

[KLR+03] D. Koller, P. Lindstrom, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. Virtual GIS: A real-time 3D geographic information system. In *Proceedings of the Conference on Visualization (VIS03)*, page 94. IEEE Press, 2003.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[MEH+99] A. MacEachren, R. Edsall, D. Haug, R. Baxter G. Otto, R. Masters, S. Fuhrmann, and L. Qian. Virtual environments for geographic visualization: Potential and challenges. *Proceedings of the ACM Workshop on New Paradigms for Invormation Visualization and Manipulation*, pages 35–40, 1999.

[Min94] M. Mine. Virtual environment interaction techniques. In *SIGGRAPH 1997 Course Notes 27*, pages B1–B21, 1994.

[MR04] C. Ma and J. Rokne. 3D seismic volume visualization. In *Integrated Image and Graphics Technologies*, pages 241–262. Kluwer Academic Publishers, 2004.

[RGWE03] S. Roettger, S. Guthe, D. Weiskopf, and T. Ertl. Smart hardware-accelerated volume rendering. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '03*, pages 231–238, 2003.

[RSH05] T. Ropinski, F. Steinicke, and K. Hinrichs. Interactive importance-driven visualization techniques for medical volume data. In *Proceedings of the 10th International Fall Workshop on Vision, Modeling, and Visualization (VMV05)*, pages 273–280, 2005.

[SEG02] SEG-Y rev 1 Data Exchange Format. Technical report, SEG Technical Standards Committee, 2002.

[SES05] N. Svakhine, D. Ebert, and D. Stredney. Illustration motifs for effective volume visualization. *Computer Graphics and Applications Special Issue 'Smart Depiction in Visual Communication'*, pages 31–39, May/June 2005.

[SGLM03] B. Salomon, M. Garber, M. C. Lin, and D. Manocha. Interactive navigation in complex environments using path planning. *Symposium on Interactive 3D Graphics*, pages 41–50, 2003.

[SML+99] T. M. Sheffield, D. Meyer, J. Lees, H. G. Kahle, B. Payne, and M. J. Zeitlin. Geovolume visualization interpretation: Color in 3-D volumes. *The Leading Edge*, pages 668–674, June 1999.

[SRH05] F. Steinicke, T. Ropinski, and K. Hinrichs. Multimodal interaction metaphors for manipulation of distant objects in immersive virtual environments. In *Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 45–48, 2005.

[VGe06] VoxelGeo. Technical report, Paradigm, 2006.

[VKG05] I. Viola, A. Kanitsar, and E. Gröller. Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2005.

[WZMK05] L. Wang, Y. Zhao, K. Mueller, and A. Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *Proceedings of IEEE Visualization (VIS) 2005*, pages 367–374, 2005.

Volume Wires : A Framework for Empirical Non-linear Deformation of Volumetric Datasets

S.J. Walton and M.W. Jones
Swansea University
cssimon@swansea.ac.uk m.w.jones@swansea.ac.uk

ABSTRACT

We introduce a new framework for non-linear, non-reconstructive deformation of volumetric datasets. Traditional techniques for deforming volumetric datasets non-linearly usually involve a reconstruction stage, where a new deformed volume is reconstructed and then sent to the renderer. Our intuitive sweep-based technique avoids the drawbacks of reconstruction by creating a small attribute field which defines the deformation, and then sending it with the original volume dataset to the rendering stage. This paper also introduces acceleration techniques aimed at giving interactive control of deformation in future implementations.

Keywords: Volume rendering, Volume deformation, Swept volumes, Curves, Volume Animation, Nonlinear deformation, Attribute distance field

1 INTRODUCTION

Research in the area of volume graphics is mainly concentrated on visualisation techniques. Tools and API's for volume modeling [SK00] and visualisation [WC01] exist, but there is a lack of tools and techniques for interactively manipulating these datasets. For surface-based graphics, a huge variety of tools exist (such as Maya and Character Studio) for the manipulation and rendering of such objects. It would be beneficial to the volume graphics community to bring some of the concepts of such powerful animation tools to working with volume datasets.

Volumetric deformation techniques have been recently documented in the literature [CCI$^+$05]. Deforming volumetric datasets is viewed as a more complex problem than surface-based deformation due to the size of the data. Even if one extracts a subset of this data (a *volume object*) with segmentation techniques [Lak00], the number of voxels to be deformed is still a limiting factor. Some approaches rely on either converting to an intermediate representation (using marching cubes to convert to a mesh structure) and then deforming that representation, or reconstructing (voxelising) a newly deformed volume dataset to be passed to the rendering stage.

This paper introduces a new software-based method to deform a volumetric dataset non-linearly without converting to a mesh geometry or using expensive volume reconstruction techniques. Our work concentrates on empirical deformation with the aim of producing a simple to use volume deformation and animation tool.

Journal of WSCG, ISSN 1213-6972, Vol. 14, 2006
Plzen, Czech Republic.

2 RELATED WORK

We split the related work into two logical areas - volume deformation and swept volumes.

2.1 Volume deformation

Spatial Transfer Functions [CSW$^+$03] were introduced by Chen *et al* . They define a framework for specifying spatial transformation and deformation for volume objects. A spatial transfer function defines the geometrical transformation of every point in the volume. Typically, a backward-mapping operation must be performed (the inverse of the deforming function) to find out where to sample in the dataset based on the current sample point on the ray. Depending on the complexity of the function, the computational cost can be high.

Similar non-reconstructive approaches involve placing ray deflectors in the scene [KY95] which deform the ray as it passes through the volume, but its use is rather limited, and specifying the deflectors is typically unintuitive as the user must think in terms of the reverse effect. Hardware-accelerated methods that work with isosurfaces exist such as in [WRS01], however, specifying the deformations is still unintuitive for the user, and isosurface property restrictions exist. Other techniques such as the 3D chainmail algorithm [Gib97] rely on moving the individual voxels and then splatting the newly-positioned voxels to the screen [Wes90]. These methods still (e.g. for animation purposes) do not allow for intuitive deformation on a large scale from the perspective of the user.

More recent work by Gagvani [GS01] has allowed for the widely-used IK-skeleton deformation methods to be utilised in volume graphics, whereby an entire new volume is reconstructed and then rendered. The algorithm is costly when the size of the dataset is large (for example, the visible human), as for the case of an animation, a new dataset must be created for each frame. A small animation can easily run over 50GB when stored on disk.

Prakash and Wu [WP99] animated the visible human using Finite Element Methods and clustering for segmenting the dataset into blocks. A hardware accelerated manipulation system called VolEdit [SSC03] allows the user to interactively manipulate the IK-skeleton and see the results in real-time. Since the transformations are linear, cracks can appear at joint areas. The VolEdit system solves this problem using mid-plane geometry. Part of the motivation for our work in this paper has been to solve this problem with a software based, non-reconstructive method.

2.2 Swept volumes

A *swept object* is produced when some template is swept along a trajectory through space. The template to be swept can be a static template such as a 2D image, or a dynamic template that changes through the sweep. Complex swept objects can be achieved by scaling [BvNP89] or rotating the template as it is swept. If the template varies as with slices through an axis of a volume dataset, then the result is a volume dataset swept along a new trajectory.

Much work has been published on swept volumes with an excellent review of techniques given in [AMBJ00]. The amount of work published is a reflection of the difficulty of some of the associated problems with sweeping techniques – in particular, the problem of determining properties of a swept object such as its boundary and volume. Early work on swept solids by Kajiya [Kaj83], and Wijk [vW84] go into some detail on methods for ray-tracing swept solids defined with arbitrary paths. Sealy and Wyvil [Sea97] describe how to voxelise new volume objects by sweeping contours along a curve, which is achieved by recursively subdividing the curve.

In [WC02], 2D images are swept along a path defined by a Bézier curve to reconstruct a volume. The volume is rendered using direct volume rendering. The authors also discuss attempts to directly evaluate the resulting deformation without reconstructing a volume, but unfortunately such evaluation is expensive (since it involves using numerical root finding methods), restrictive, and problematic (e.g, singularity conditions on an axis where an image is swept around the axis).

A *swept volume* is produced when a swept object is voxelised [Sea97]. The new volume can then be rendered using any volume visualisation technique. The disadvantage of reconstructing a volume from a sweep is the space requirement – a new volume must be produced and either stored in memory or on disk. For an animation, this is multiplied by the number of frames if the user wishes to retain the intermediate data to re-render the animation at a later date, with new view parameters or lookup functions.

3 DISTANCE FIELDS AND ATTRIBUTE PROPAGATION

Since a distance field technique is required for our method, we present a brief overview. Distance fields [SJ01] have been widely used for a variety of applications in the volume domain, such as morphing [BW01], voxelisation [Jon96] [JS00], and skeletonisation [GS01]. A distance field dataset D representing a surface S is defined as $D : \mathbb{R}^3 \to \mathbb{R}$, and for $p \in \mathbb{R}^3$,

$$D(p) = min\{\mid p - q \mid : q \in S\} \tag{1}$$

where $||$ is the Euclidean norm and q are the nearest points on the surface. Each voxel in the field contains a value that represents the minimum distance to the surface of interest in the data. In the case of volume data, we may be interested in a particular isosurface representing, for example, the bone surface in a medical dataset. We can sign the value depending on whether the voxel is inside the target surface - becoming a signed distance field. A fast method of computing this field is by using the distance transforms [SJ01] to propagate local distances.

It follows that if we can propagate the minimum distances to a surface in this way, any related attributes of the surface (e.g. colour, as in [BM99]) can also be propagated. These additional attributes can be stored at each voxel in the distance field. The field then becomes:

$$D(p) = (min\{\mid p - q \mid : q \in S\}, a_1, \ldots, a_n) \tag{2}$$

where $a_1, \ldots, a_n$ are our additional attributes. If only the attributes are of interest then the distance value at each voxel may be discarded, thus saving typically 4 bytes per voxel if using floating-point precision. In our method, we discard the distance values as they are not needed in later stages.

4 METHOD DESCRIPTION

Our approach is based on the idea of sweeping a volume object along an arbitrarily defined path, although the approach may also be viewed from the standpoint that the deformed path has the effect of deforming the surrounding volume object. Because no reconstruction takes place, the deformation and rendering stages are closely coupled. Figure 1 gives a high-level overview of the system. The method is not limited to specific classes of curve or any other trajectory definitions, except for the requirement that it can be parametrically evaluated, satisfying the general form:

$$\alpha(t) = (\alpha_x(t), \alpha_y(t), \alpha_z(t)) \tag{3}$$

The deformed dataset is evaluated at render time using an *attribute field*, and can be rendered easily with a ray-casting renderer using backward-mapping operations.

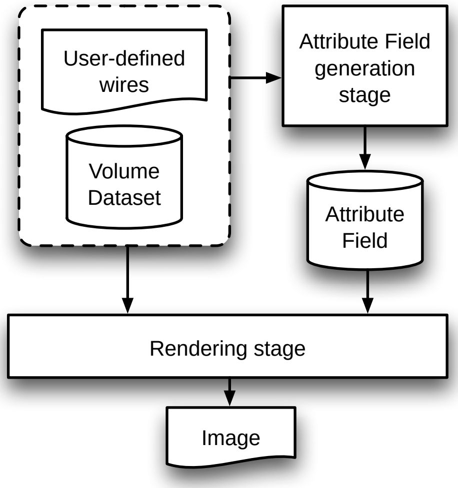
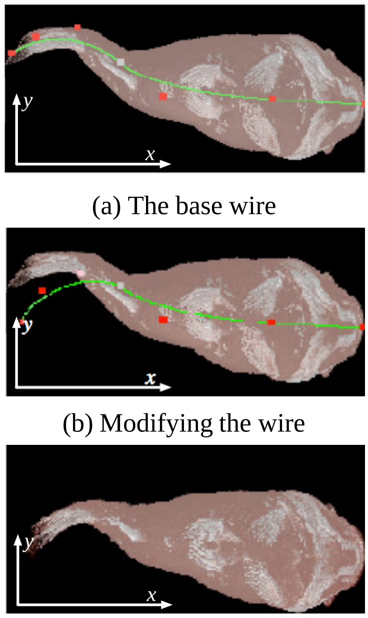


Figure 1: System overview



(a) The base wire

(b) Modifying the wire

(c) The resulting render

Figure 2: Fish deformation

4.1 Specifying the deformation

From the user's point of view, the specification of the deformation is conceptually similar to that of wire deformers in Maya. We are therefore extending this surface-based deformation technique to work neatly with volumetric datasets. Such a transition is not trivial due to the entirely different data representation (discretely sampled vs. surface based). In addition, an important difference is that we are not deforming the data itself and sending it to the rendering stage.

In our method, the user defines a *base wire* close to or inside the object to be deformed, and also an object classification function for the wire $\beta(p \in \mathbb{R}^3) \rightarrow [true, false]$ which determines the associated volume object. The wire is then transformed via translations, rotations, and curve deformation and this has the effect of deforming the volume object defined by β in the wire's specified region of influence.

The method permits scaling and rotation values at arbitrary points on the wire. For example, an angle of $\theta = 0$ at one end of the wire and $\theta = \pi$ at the other will produce the effect of the object being twisted along the path (linearly interpolating the θ values). The length of the base wire and the modified wire need not be equal, which allows for compression and expansion of the data along the trajectory of the wire.

Figure 2(a) gives an overhead view of the user-defined base wire on the CT carp dataset. Figure 2(b) shows that the user has modified the wire, pulling one end of the wire in the negative y-axis direction. Finally, Figure 2(c) shows the resulting render from this deformation. In this example, the wires are Bézier curves. The base wire in this instance acts as the backbone of the carp. Deforming this backbone and then rendering the result would result in a new pose for the carp, as the surrounding soft tissue would be deformed around the backbone. The backbone could be derived semi-automatically using a simplified distance field thinning technique [GS01] (choosing the strongest segment) or watershed segmentation technique [Lak00].

5 BUILDING DEFORMATION DATA

In this stage, the deformation information from the wire-specification stage is encoded into an *attribute field*, which is then sent with the original volume object to the rendering stage. The attribute field is a volumetric dataset (γ, ε) where γ maps voxels to their corresponding wire and ε maps voxels to the t-value (see equation 3) of the closest point on that wire. For certain classes of curve (e.g. Catmull-Rom splines), the segment index also needs to be stored.

The attribute field need not be the same scale as the volume. In our research we have found that producing an attribute field of 1/8th size (half each dimension) produces results very close visually to the full size field. Additional considerations regarding reduced scale fields and example images are given in a later section.

For each base wire, we associate a set of planes P (see Figure 3) aligned with the trajectory of the wire. The plane dimensions are automatically defined to tightly fit the target object defined by classification function β within the wire's region of influence. To generate the attribute field, an empty field is initialised over the domain of the union of each of the wires' region of in-

ISBN 1213-6972 ISBN 80-86943-09-7

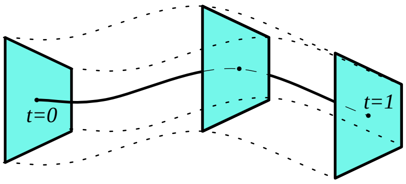


Figure 3: Planes defined along wire

fluence. A mapping is now defined between planes on the base wire and planes on the modified wire, essentially the planes are copied based on their t-value. For each plane on the modified wire, we look at the attribute field voxels touched by the plane. If the plane touches a voxel, then a flag is set with that voxel.

The optimal number of planes can be calculated from an approximation of the wire's length. For parameterised curves, the length can be approximated with precision p by:

$$|\alpha| = \sum_{i=2}^{p} |\ \alpha(\frac{i}{p}) - \alpha(\frac{i-1}{p})\ | \qquad (4)$$

where $||$ is the Euclidean norm.

5.1 Voxel Initialisation

Each wire is now voxelised into the attribute field. When a new cell is entered by the wire, each of the eight surrounding voxels' γ (the wire reference) and ε (closest t-value) attributes are set, and also the distance d from the voxel to the point defined by ε, as shown in Figure 4. The closest point on the wire is calculated by subdividing the subset of the curve inside the voxel (as in equation 4), and this calculation can be achieved with parametrically-defined precision[1]. If a voxel has already been set in a previous cube (as with v_4), then the new and current minimum distances are compared and the minimum taken. This is denoted by the greyed-out vector in Figure 4 where $d' < d$.

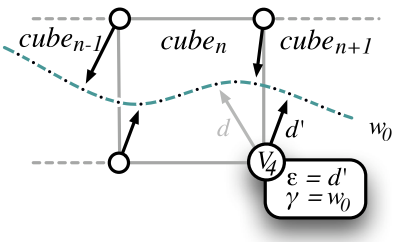
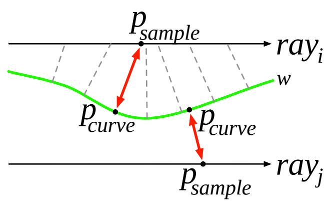


Figure 4: Pre-propagation voxel initialisation

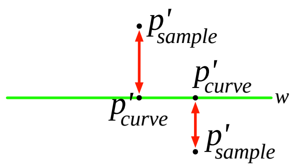
Once this process is complete for all wires, the distances and associated attributes are propagated using a distance transform method, and the distance values are discarded. The propagation only takes place with voxels flagged in the previous step, so large areas of the field can be skipped. It is this propagation that removes the need for a costly backward evaluation at each voxel.

6 RENDERING THE DEFORMATION

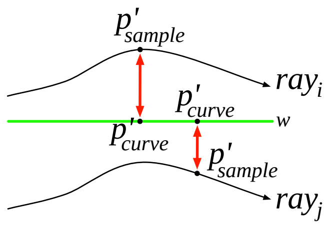
To render the deformation, a standard ray-casting approach is followed, with rays cast into the attribute field instead of the volume object. We choose a ray-casting approach to ensure a possible straightforward GPU implementation for Geforce 6 based cards. All voxels in the field which have not been flagged are ignored. At each sample point p_{sample} on the ray, the wire reference w from the nearest voxel is noted. For the current sample point p_{sample}, the wire parameter value t is trilinearly interpolated from the eight surrounding voxels. The mapping achieved between the base wire and



(a) The modified wire



(b) The base wire



(b) The effective path of rays i and j

Figure 5: The mapping between wires

the deformed wire is illustrated in Figure 5. Given the wire reference w and t-value t, we can calculate the closest point on the modified wire and build a vector to it, becoming $p_{sample} \rightarrow p_{curve}$. To obtain the actual sample point in the volume dataset from this, vector $p_{sample} \rightarrow p_{curve}$ is mapped onto the base wire, becoming $p'_{sample} \rightarrow p'_{curve}$ by using the t-value. This is demonstrated in Figure 5 where two sample points on ray_i and ray_j are mapped from the modified wire (a) to the base wire (b). p'_{sample} is now our new sample point

[1] This is a fairly fast and accurate way to approximate the closest point on a curve. Spline implicitization [Sha03] or other methods [Sch90] could be used if more precision is required, at the expense of additional complexity.

in the dataset. The final effect of ray_i and ray_j's trajectories being deformed is shown in (c).

If the attribute field has been scaled with respect to the volume object, then the density of sample points in the field must be modified accordingly. We also must deal with cases where a cube's eight vertices give different wire references, as in Figure 6. The interpolated t-value at p_{sample} would be inaccurate for either choice of wire (a or b). We have looked at fast methods for recovering a t-value (such as taking averages of the majority wire), but we have found that the decision at these voxels contributes little to the final image quality except with very low scale fields. In the resulting images (given later), we simply choose the wire and t-value at the closest voxel to p_{sample}.

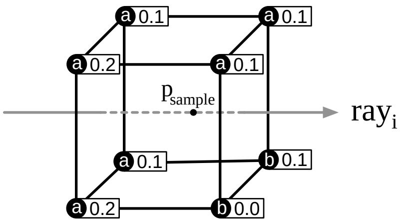


Figure 6: Differing wire reference problem

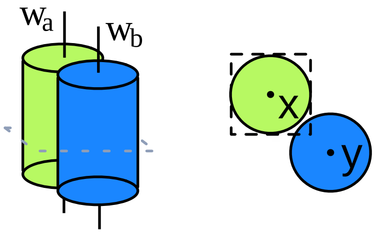
6.1 Calculating the new normals

Once the the new sample point has been calculated, a new normal at that point is required if we are to accurately light the deformed object. One way to achieve this would be to use central differences using backward-mapped points, but this is clearly an expensive operation. Therefore, to compute the deformed normal, we first compute the normal n at the new sample point p'_{sample} obtained in the backward-mapping stage. This normal can be calculated using central differences in the original volume dataset at p'_{sample}. To obtain a new normal n', we transform n by the inverse of the backward-mapping transformation obtained for the current sample point. n' is then sent to a lighting equation.

7 OPTIMISATION AND THE DELINEATION PROBLEM

Problems may arise when the user wishes to deform two objects that are in close proximity – perhaps by pulling the two objects apart to separate them. We illustrate this problem in Figure 7. Figure 7(a) shows two objects x and y, and Figure 7(b) shows a slice of the objects (the slice is shown half-way down the objects in (a)). In this case, a plane of target object x's wire (shown as a dotted rectangle) has overlapped object y. Part of object y will therefore be included in the deformation of object x, since y is within x's plane. This is unlikely to be what the user would have intended in this case.

If the user has defined multiple wires inside the volume dataset, it is likely that they wish to treat the dataset as a set of disjoint volume objects as defined by function β. It would be favourable for the system to be able to automatically delineate the objects in Figure 7 without the user resorting to volume segmentation methods [Lak00], which are typically very difficult to work with.



(a) (b)

Figure 7: The delineation problem

7.1 Plane masks

To solve the problem discussed above, *plane masks* are introduced. Once the planes are defined on the wire, a 2.5D seed fill[2] is performed on each of the planes to generate a 2D bit-mask, which is then stored with the plane.

Figure 8 shows a selection of these masks defined along the wire for the CT carp dataset, with an object classification function β set to identify the outer skin area with a simple value threshold. The resolution of the mask can be varied by parameter s, and the memory requirement for each wire in bytes is calculated as:

$$\sum_{p=1}^{n} \frac{(a(p) * s)}{8}$$

where n is the number of planes on the wire, a gives the area of the plane, and s is a resolution scale multiplier. Values of s below 1 give a sparse mask, values above give a fine mask and therefore greater precision, at the expense of a greater storage overhead and preprocessing time.

Figure 8: Masks defined along wire

The algorithm automatically hunts for an appropriate seed point by searching inside the plane area outwards from the wire. The condition for a fill at each pixel in the bit-mask is the wire's β function. If a suitable seed point is not found, then the plane is removed from the

[2] Essentially, a 2D image cutting through the volume dataset - the 2D bit-mask is filled, and the part of the volume touched by the plane used to identify the target object.

list, as no object data has been found within the plane's subsection of the volume. To ensure that data at the edges is not skipped, we also apply a morphological dilation operation to the mask. Voxels in the attribute field are now only flagged if the plane mask bit at that point is 1 (See Figure 9).

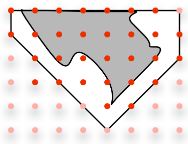


Figure 9: A Plane mask flagging voxels

This solution is effective in that not only does it solve the delineation problem, but it also further reduces the number of flagged voxels in the scene, which reduces rendering time. Backward-mapping operations are now only performed on voxels whose resulting new sample positions lie within the target object (or slightly out). Table 1 gives the number of non-flagged voxels ignored for some example deformations. Note that we do not include samples outside of the field boundaries in the figures.

Dataset	# Sample pts	# Pts ignored	% ignored
CT Carp	26,011,195	15,523,566	59.7%
Visman	45,461,270	39,253,862	86.3%

Table 1: Voxels ignored while rendering

7.2 Speed / Storage / Accuracy trade-offs

Each voxel in the attribute field requires three attributes. The first is γ : the wire reference, the second is ε : the t-value on the wire, and the third is a bit for the flag that denotes a voxel has been swept. If we assume floating-point precision on the t-value, we have a minimum of 5 bytes per voxel including 7 bits for the wire reference with a maximum of 128 wires in the scene. This storage requirement can be reduced by using integer precision on the t-value. Below is an example 2-byte per voxel solution for Catmull-Rom spline wires.

# Bits	Range of values	Data
1	2	swept flag
4	16	w : wire reference
4	16	s : segment index on w
7	128	integer t-value on s

The integer precision on the t-value has another advantage. The points at each integer offset on the wire can be cached before the rendering stage and then used during rendering to avoid expensive curve evaluation at each sample point (the points are chosen by subdividing the curve as in equation 4). The wire point calculation is now reduced to a simple array lookup. If we wish for greater precision still, linear interpolation can be performed between values. The same technique can be used for the wire normals : for each modified wire point p, the difference between the wire normal at p and the normal at the same t-value on the base wire is calculated, and stored.

8 IMPLEMENTATION

The method has been implemented in C++ on GNU / Linux x86. To assist with rapid testing, and to demonstrate the simplicity of specifying deformations, we have built a simple user interface using the GTK+ library. The interface allows the user to view the volume dataset from multiple angles interactively and quickly define and deform wires. The user can also specify an animation by deforming the wire differently for an arbitrary number of frames. The wires can be saved to disk for later retrieval, and rendered into a series of images which can be encoded into a movie.

9 RESULTS

To give a more accurate representation of the overhead of our method implementation, we first give the timings for a software ray-casting volume rendering algorithm written in C++ with very few optimisations (see table 2), and then modify it to work with our method (results in table 3). *Preprocessing* refers to the attribute field generation stage, which also includes mask generation, curve lookup table generation, and other pre-render data discussed in previous sections. The difference in the timings gives the overhead of calculating the attribute field and transforming sample positions in each case.

The base wire and deformed wires are identical to give the same number of sample positions during volume rendering (thus ensuring a fair comparison). The deformation is therefore the identity deformation. The viewing parameters and image size of 512x256 are also constant. The timings are based on a P4 at 3.2GHz with 512MB RAM.

Dataset	Render time
CT Carp	5.74 secs
Tubes	2.59 secs
Visman torso	4.31 secs

Table 2: Standard rendering times

Dataset	Preprocessing	Render	Total
CT Carp	1.78	12.65	14.43
Tubes	0.96	4.81	5.77
Visman torso	2.97	22.87	25.84

Table 3: Deform/render times

The timings were performed using all acceleration techniques discussed, but the majority of code has not

 ISBN 1213-6972 ISBN 80-86943-09-7

yet been optimised. The tables show that the overhead in the rendering stage is far higher than the preprocessing stage. The biggest factor in the cost of attribute field generation is the size of the field, as more propagation must take place.

Figure 12 shows the visible human rendered with the same deformation (the head has been pulled back), with differing attribute field to dataset ratios. 1:1 (same dimensions) predictably gives the most pleasing reproduction, while a 1:4096 (each dimension is 1/16th the size) field gives a blocky appearance due to trilinear interpolation taking place in the large gaps between voxels. We also give the time for attribute field generation for each.

10 CONCLUSION

We have introduced a new software-based framework for non-linear, non-reconstructive deformation of volumetric datasets. The framework brings a much-needed intuitive deformation method to the field of discretely sampled object representations. The lack of such methods available for volume deformation severely hampers the area, and we feel that this framework goes some way to correct this.

We have shown that the method requires only a small memory storage overhead, and avoids the discussed disadvantages of reconstruction-based methods. The specification of such deformations can be easily defined without knowledge of the internal algorithms that deform the data. The problem of delineating volume objects to deform independently is also handled in a simple manner.

(a) ratio 1:1, time 11.74s

(b) ratio 1:8 (1:2 dim), time 4.45

(c) ratio 1:64 (1:4 dim), time 2.61s

(d) ratio 1:512 (1:8 dim), time 2.17s

(e) ratio 1:4096 (1:16 dim), time 2.12s

Figure 12: Attribute field scales

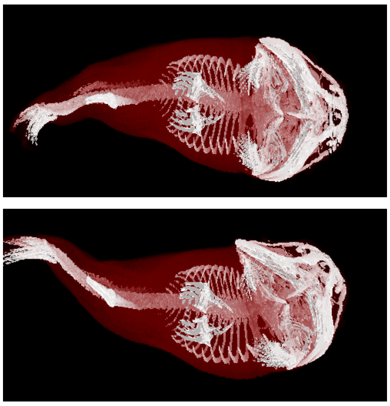


Figure 10: CT Carp deformation

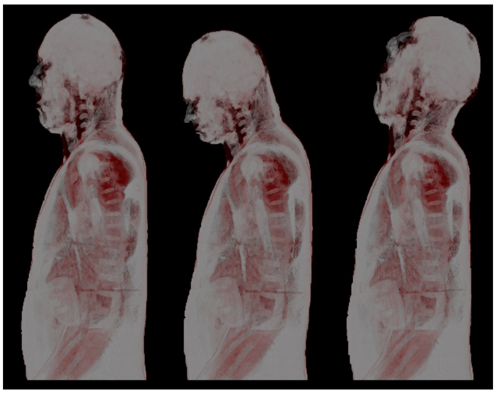


Figure 11: Visible human deformation

In addition, the standard ray-casting approach to volume rendering can be used to render the result with only minor modifications to the rendering engine. This facilitates the method's integration into the volume deformation and rendering pipeline. We have recently implemented the rendering stage on the GPU by loading the attribute field as a 3D texture, so the total time required is now little more than the field generation overhead.

11 ACKNOWLEDGEMENTS

This work has been supported by EPSRC grant GR / S44198. The authors would also like to acknowledge

Stefan Roettger's volume library [Roe], and the National Library of Medicine's Visible Human project.

REFERENCES

[AMBJ00] K. Adbel-Malek, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. In *International Journal of Shape Modeling*, 2000.

[BM99] D.E. Breen and S. Mauch. Generating shaded offset surfaces with distance, closest-point and color volumes. In *Proceedings of the International Workshop on Volume Graphics*, pages 307–320, March 1999.

[BvNP89] W. F. Bronsvoort, P. R. van Nieuwenhuizen, and F. H. Post. Display of profiled sweep objects. *The Visual Computer*, 5(3):147–157, 1989.

[BW01] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):173–192, 2001.

[CCI+05] M. Chen, C Correa, S Islam, M. W. Jones, P.Y. Shen, D Silver, S. J. Walton, and P. J. Willis. Deforming and animating discretely sampled object representations. In *Eurographics 2005 STAR Reports*, pages 113–140, Dublin, Ireland, August 2005.

[CSW+03] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea. Spatial transfer functions – a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics 2003*, pages 35–44, Tokyo, Japan, 2003.

[Gib97] S. Gibson. 3D chainmail: a fast algorithm for deforming volumetric objects. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 149–154, April 1997.

[GS01] N. Gagvani and D. Silver. Animating volumetric models. *Graphical Models*, 63(6):443–458, 2001.

[Jon96] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[JS00] M.W. Jones and R.A. Satherley. Shape representation using space filled sub-voxel distance fields. In *Vision, Modeling and Visualization*, pages 316–325, 2000.

[Kaj83] J.T. Kajiya. New techniques for ray tracing procedurally defined objects. In *SIGGRAPH '83*, pages 91–102, New York, NY, USA, 1983.

[KY95] Y. Kurzion and R. Yagel. Space deformation using ray deflectors. In *Proc. 6th Eurographics Workshop on Rendering 1995*, pages 21–32, Dublin, Ireland, June 1995.

[Lak00] S. Lakare. 3D segmentation techniques for medical volumes. http://www.cs.sunysb.edu/ mueller / teaching / cse616 / sarangRPE.pdf, 2000.

[Roe] Stefan Roettger. The volume library. http://www9.cs.fau.de / Persons / Roettger / library/.

[Sch90] P. Schneider. Solving the nearest-point-on-curve problem. In *Graphics Gems*, volume 1, pages 607–612. Academic Press, 1990.

[Sea97] G. Sealy. Representing and rendering sweep objects using volume models. In *CGI '97*, pages 22–27, Washington, DC, USA, 1997.

[Sha03] M. Shalaby. Spline implicitization of planar curves and applications. *PhD thesis, Johannes Kepler University*, 2003.

[SJ01] R.A. Satherley and M.W. Jones. Vector-city vector distance transform. *Computer Vision and Image Understanding*, 82(3):238–254, 2001.

[SK00] M. Sramek and A.E. Kaufman. vxt : A c++ class library for object voxelisation. In *Volume Graphics*. Springer, 2000.

[SSC03] V. Singh, D. Silver, and N. Cornea. Real-time volume manipulation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 45–52, 2003.

[vW84] Jarke J. van Wijk. Ray tracing objects defined by sweeping planar cubic splines. *ACM Trans. Graph.*, 3(3):223–237, 1984.

[WC01] A.S. Winter and M. Chen. *vlib*: A volume graphics API. In *Volume Graphics 2001*. Springer-Wien New York, 2001.

[WC02] A.S. Winter and M. Chen. Image-swept volumes. *Computer Graphics Forum*, 21(3):441–441, 2002.

[Wes90] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[WP99] Z. Wu and E.C. Prakash. Visible human walk: bringing life back to the dead body. In *VG99*, pages 347–356, 1999.

[WRS01] R. Westermann and C. Rezk-Salama. Real-time volume deformations. In *Comput. Graph. Forum*, volume 20, 2001.

Detecting Holes in Point Set Surfaces

Gerhard H. Bendels Ruwen Schnabel Reinhard Klein

Universität Bonn
Institut für Informatik II – Computergraphik
Römerstraße 164
D-53117 Bonn, Germany

ABSTRACT

Models of non-trivial objects resulting from a 3d data acquisition process (e.g. Laser Range Scanning) often contain holes due to occlusion, reflectance or transparency. As point set surfaces are unstructured surface representations with no adjacency or connectivity information, defining and detecting holes is a non-trivial task. In this paper we investigate properties of point sets to derive criteria for automatic hole detection. For each point, we combine several criteria into an integrated boundary probability. A final boundary loop extraction step uses this probability and exploits additional coherence properties of the boundary to derive a robust and automatic hole detection algorithm.

Keywords Point Set Surfaces, Modelling, Filtering, Repairing

1 Introduction

Point set surfaces have become popular with the rise of 3D data acquisition techniques such as laser-range scanning. Their conceptual simplicity makes them suitable for both modelling as well as high quality rendering. Usually, these 3D data acquisition methods deliver unstructured point clouds, possibly equipped with normals and additional surface properties, such as colour. The surface is encoded implicitly therein and can only be extracted using some neighbourhood relation between samples. Compared to mesh based representations, the lack of explicit connectivity information simplifies the definition and implementation of many tasks encountered in geometric modelling, such that for instance free-form deformation techniques for point sets become increasingly popular [PKKG03, BK05]. On the other hand, the detection of holes in the surface – trivial in the case of meshes – becomes an ill-defined problem.

The knowledge of holes in the data, however, is vital for many applications dealing with point set surfaces and it can be exploited in several ways. It can be used to reconstruct surfaces with boundaries or to direct a further scanning step, gathering missing information in holes, either manually or even automatically. In postprocessing, a smoothing step to remove noise profits from boundary information as many smoothing operators usually fail on boundaries and special handling is required at the borders. Identification of points on the boundary of a hole is obviously required before any attempt to algorithmically fill holes, an application useful not only in surface repairing but also in modelling and interactive editing [BSK05, SACO04].

While several authors proposed sampling conditions for surfaces to ensure correct reconstruction (most notably [ABE98]), we are not primarily concerned with undersampling but are interested in holes that a human user might identify when inspecting a point cloud, often unaware of the original surface. Also we want to provide a user with intuitive parameters making it easy to find the holes needed for a given application.

The Journal of WSCG, Vol.14, ISSN 1213-6972
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2 Previous Work

The problem of detecting holes in point set surfaces is closely related to surface reconstruction as well as feature extraction. Thus, many algorithms in those areas include criteria to identify holes or undersampled surface patches.

[GWM01], [LP02] as well as [CN04] apply what we shall call the angle criterion. The angle criterion considers for each sample point $\mathbf{p}$ a set of neighbouring samples and examines the maximum angle between two consecutive neighbours. [GWM01] also use the

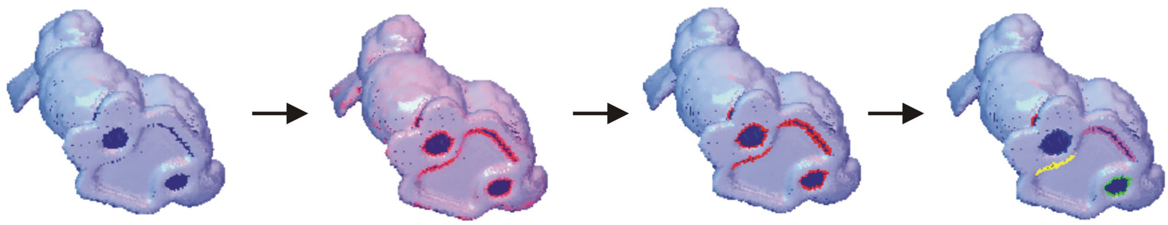


Figure 1: The steps of the boundary detection algorithm. From left to right: A boundary probability $\Pi(\mathbf{p})$ is computed for every point (the points are shaded in red according to their boundary probability). Then points are classified into boundary and interior points, exploiting coherence. Finally, for each hole a boundary loop is extracted.

correlation matrix formed by the neighbourhood. The eigenvectors and eigenvalues of this matrix define a correlation ellipsoid. Its shape, expressed in the ratios of the eigenvalues, is used to identify corner, crease and boundary points and also gives an approximation to crease and boundary direction. In order to find continuous crease lines, a neighbourhood graph on the point set is built and its edges are weighted according to the crease probability. Edges with high probability are then collected and constitute the feature patterns.

In [DG01], undersampled regions are detected using the sampling requirement of [ABE98]. This sampling condition is based on an approximation of the medial axis by so called poles of each sample's Voronoi cell. The distance of each point to the medial axis gives the local feature size. Every point on the true surface needs at least one sample point within a ball defined by the local feature size and a factor r. Consequently, [DG01]'s approach fails to identify holes in flat areas of the surface, where only very few samples are required to fulfill this requirement (in flat areas the medial axis is far away). In these areas, though, often holes are present and clearly visible for a human observer. Similarly, we are not interested in regions declared undersampled at sharp creases where the sampling requirement can never be met (at sharp edges the medial axis touches the surface).

3 Overview

Let $\mathcal{S}$ be a 2-manifold surface and let the set of points $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathbb{R}^3$ be a (not necessarily regular) sampling of $\mathcal{S}$. Suppose also that $\mathbf{n}_1, \ldots, \mathbf{n}_n$ are the corresponding surface normals. The problem is now to define an operator

$$B_{\mathcal{P}} : \mathcal{P} \to 2^{\mathcal{P}} \ ; \ B_{\mathcal{P}}(\mathcal{P}) \mapsto \{\mathbf{p} \in \mathcal{P} | \mathbf{p} \text{ is boundary}\}$$

that identifies the set of boundary points $\mathcal{B} = B_{\mathcal{P}}(\mathcal{P})$ circumscribing holes in $\mathcal{P}$. We denote the boundary operator with a subscript $\mathcal{P}$ to stress that the assignment *boundary* or *non-boundary* is strictly a property of the point set under consideration itself.

The basic layout of our hole detection scheme (depicted in figure 1) is as follows: For each point $\mathbf{p} \in \mathcal{P}$ we compute a boundary probability $\Pi(\mathbf{p})$, reflecting the probability that $\mathbf{p}$ is located on or near a hole in the surface sampling (section 4). Thereafter, we exploit that the boundary property is coherent, i.e. that boundary points have proximate neighbours that are also boundary, and construct closed loops circumscribing the hole in a shortest cost path manner (section 5). Results and applications of our hole detection scheme are given in sec. 6.

4 Boundary Probability

The property of *being boundary* inherently is a property of the local neighbourhood of $\mathbf{p}$ rather than of the point $\mathbf{p}$ itself. In order to define and evaluate the boundary criteria, we therefore have to seize the local neighbourhood $N_{\mathbf{p}}$ more formally.

4.1 Neighbourhood Collection

A very common definition of local neighbourhoods around a point $\mathbf{p}$ found in the literature is the k-neighbourhood $N^k_{\mathbf{p}}$, consisting of the k nearest samples in $\mathcal{P}$ to $\mathbf{p}$. This simple definition, though, becomes unreliable in areas of varying sampling density. In points lying on the edge of a densely sampled region, the k-neighbourhood will be biased towards the densely sampled region (fig. 2, left).

This problem can be alleviated somewhat by the $N^{k\epsilon}_{\mathbf{p}}$ neighbourhood, that includes not only the k nearest points but also all points inside a small sphere with radius ϵ. By selecting an appropriate value for ϵ, the biasing effect can be reduced, but the neighbourhood of points in densely sampled regions will contain more points than necessary, increasing the cost of evaluating the boundary criteria, which effectively limits the range of a feasible ϵ. For sharp sampling density

 ISBN 1213-6972 ISBN 80-86943-09-7

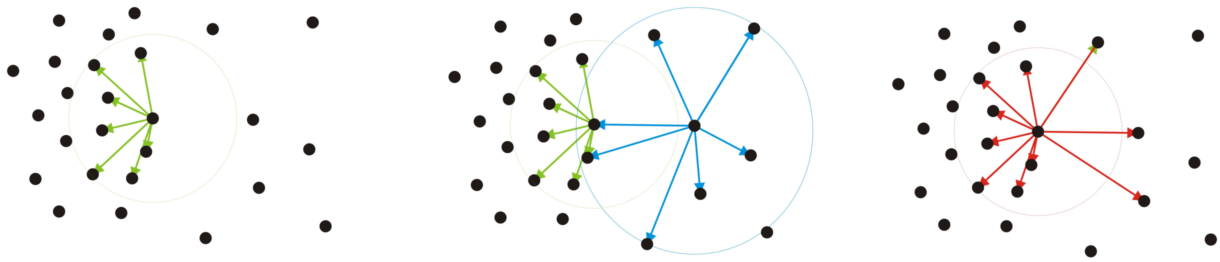


Figure 2: Left: The $k\epsilon$-neighbourhood is biased towards densely sampled regions. Middle: $k\epsilon$-neighbourhoods of points in the sparsely sampled region contain points of the densely sampled area. Right: The symmetric $k\epsilon$-neighbourhood is not affected by the change in sampling density.

changes (as often encountered in point sets stemming from registered range images), this alleviation alone is not sufficient.
However, whereas the $k\epsilon$-neighbourhood for points situated on a sampling density drop will include only points in the densely sampled region, these points will be contained in the neighbourhood of nearby points, located in the sparsely sampled region (fig. 2, middle). To overcome the biasing effect, it therefore typically suffices to include these nearby points in the neighbourhood (fig. 2, right). To complete the neighbourhood for the critical points, we hence define:

$$N_{\mathbf{p}} = \{\mathbf{q} \in \mathcal{P} \mid \mathbf{q} \in N_{\mathbf{p}}^{k\epsilon} \vee \mathbf{p} \in N_{\mathbf{q}}^{k\epsilon}\},$$

i.e. $\mathbf{q}$ is considered one of $\mathbf{p}$'s neighbours, already if $\mathbf{p}$ is one of $\mathbf{q}$'s.
To efficiently find the neighbourhood for each point, a kd-tree is built, containing all points in $\mathcal{P}$. The kd-tree supports the collection of the k nearest neighbours to a point in $O(klog^3|\mathcal{P}|)$ and can also be used to quickly retrieve all points in a sphere of radius ϵ. After the kd-tree has been constructed, we build the *proximity graph* $G(\mathcal{P}, \mathcal{E})$, with $\mathcal{P}$ as vertices and edges

$$\mathcal{E} = \{(i, j) \mid \mathbf{p}_j \in N_{\mathbf{p}_i}\}.$$

Please note that this graph is symmetric, and the adjacency lists of the graph correspond to the $N_{\mathbf{p}}$-neighbourhood of each point.

4.2 The Angle Criterion

The angle criterion projects all neighbouring points contained in $N_{\mathbf{p}}$ on the tangent plane and sorts them according to their angle around the centre sample, see figure 3, and computes the largest gap g between two consecutive projected neighbours. The basic idea is that g will be significantly larger for a boundary point than for an interior point, as illustrated in figure 3. Consequently, the boundary probability is given as

$$\Pi_{\angle}(\mathbf{p}) = \min\left(\frac{g - \frac{2\pi}{|N_{\mathbf{p}}|}}{\pi - \frac{2\pi}{|N_{\mathbf{p}}|}}, 1\right).$$

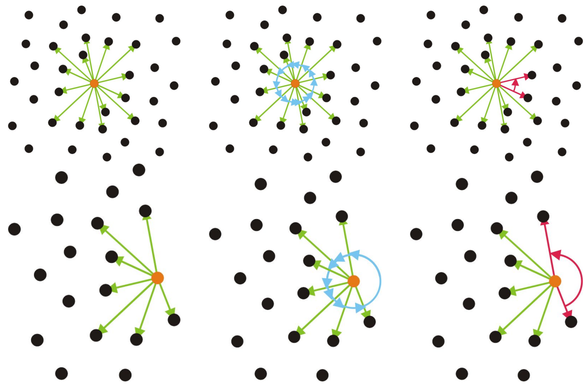


Figure 3: The three steps in the evaluation of the angle criterion for an interior point (top row) and a boundary point (bottom row). After projection into the tangent plane the difference vectors are generated (left). The projected points are sorted according to their angle around $\mathbf{p}$ (middle). The largest angular gap between two consecutive points is used to compute the boundary probability (right).

In contrast to the standard angle criterion, we ignore points $\mathbf{q} \in N_{\mathbf{p}}$ with a small scalar product $\langle \mathbf{n_p}, \mathbf{q}-\mathbf{p}\rangle$. This way the angle criterion becomes less susceptible to small inaccuracies in normal direction.

4.3 The Halfdisc Criterion

In 2D-image processing, edge detection algorithms identify pixels, whose luminance deviates considerably from the average luminance of its neighbouring pixels. The same rationale can also be applied in our problem setting. On a 2-manifold, the neighbourhood of points in the interior of the surface is homeomorphic to a disc such that we can expect the difference between the point $\mathbf{p}$ itself and the average $\mu_{\mathbf{p}}$ of its neighbours to be small, as opposed to points on a boundary. Their neighbourhood is homeomorphic to a halfdisc (see figure 4), such that $\mu_{\mathbf{p}}$ will deviate from $\mathbf{p}$ significantly. Therefore, to derive a boundary probability, we compare $\mu_{\mathbf{p}}$ with the centre of mass of an ideal halfdisc in the tangent plane. To reduce the influence

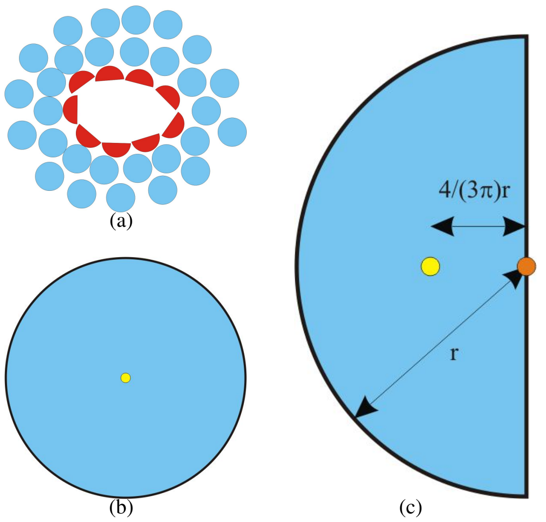


Figure 4: (a) The local neighbourhood of points located on the surface boundary is homeomorphic to a halfdisc as opposed to the full disc of an interior point. (b) For an interior point, the average of the neighbourhood points will coincide with the interior point itself, while for a boundary point (c), it will deviate in direction of the interior surface.

of variations in the sampling density, we compute $\mu_\mathbf{p}$ as a *weighted* average of $N_\mathbf{p}$ using the Gauss kernel

$$g_\sigma(d) = \exp\left(\frac{-d^2}{\sigma^2}\right),$$

where σ depends on the average distance to the neighbouring points $r_\mathbf{p}$ (namely $\sigma = \frac{1}{3}r_\mathbf{p}$), such that the influence of points outside the neighbourhood $N_\mathbf{p}$ can be neglected. This delivers:

$$\mu_\mathbf{p} = \frac{\sum_{\mathbf{q}\in N_\mathbf{p}} g_\sigma(\|\mathbf{q}-\mathbf{p}\|)\mathbf{q}}{\sum_{\mathbf{q}\in N_\mathbf{p}} g_\sigma(\|\mathbf{q}-\mathbf{p}\|)}.$$

To filter out properties of the surface itself and to include in our criterion properties of the sampling itself only, we compute the projection $\overline{\mu}_\mathbf{p}$ of $\mu_\mathbf{p}$ into the tangent plane and define the boundary probability as

$$\Pi_\mu(\mathbf{p}) = \min\left(\frac{\|\mathbf{p}-(\overline{\mu}_\mathbf{p})\|}{\frac{4}{3\pi}r_\mathbf{p}}, 1\right).$$

4.4 The Shape Criterion

As noted in [GWM01], the shape of the correlation ellipsoid of $N_\mathbf{p}$ approximates the general form of the neighbouring points. The shape of the ellipsoid is encoded in the eigenvalues $\lambda_0 \geq \lambda_1 \geq \lambda_2$ of the weighted covariance matrix $C_\mathbf{p}$:

$$C_\mathbf{p} = \sum_{\mathbf{q}\in N_\mathbf{p}} w(\mathbf{q})(\mu_\mathbf{p}-\mathbf{q})(\mu_\mathbf{p}-\mathbf{q})^t$$

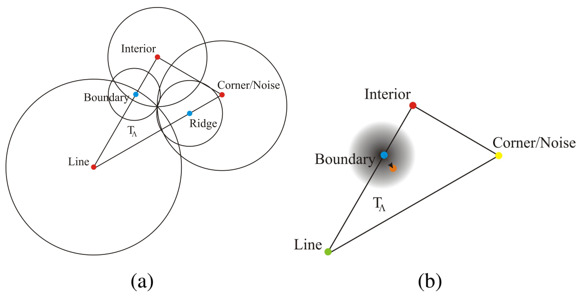


Figure 5: (a) The triangle formed by all Λ values with highlighted characteristic points for certain shapes. The circles passing through the triangle centroid **c** are shown for every shape in the respective colour. (b) The tentative probability $\widetilde{\Pi}_{boundary}$ is computed by evaluating the kernel around the characteristic Λ for boundary points.

We collect the relative magnitudes of the eigenvalues in a decision vector $\Lambda_\mathbf{p} = (\frac{\lambda_0}{\alpha}, \frac{\lambda_1}{\alpha}, \frac{\lambda_2}{\alpha})$, with $\alpha = \lambda_0 + \lambda_1 + \lambda_2$. There are four characteristic situations $\phi \in \Phi = \{$*Boundary*, *Interior*, *Corner/Noise*, *Line*$\}$:

$\phi =$ Boundary	$\Lambda_\phi = (\frac{2}{3}, \frac{1}{3}, 0)$
$\phi =$ Interior	$\Lambda_\phi = (\frac{1}{2}, \frac{1}{2}, 0)$
$\phi =$ Corner/Noise	$\Lambda_\phi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
$\phi =$ Line	$\Lambda_\phi = (1, 0, 0)$

The latter three values of Λ span a triangle T_Λ containing all possible values for Λ (fig. 5). We can now extract tentative classification probabilities $\widetilde{\Pi}_\phi$ for each of the situations described above from $\Lambda_\mathbf{p}$ by evaluating a spatial kernel around the characteristic Λ_ϕ. Again, we use a Gauss kernel g_σ with $\sigma_\phi = \frac{1}{3}\|\Lambda_\phi - centroid(T_\Lambda)\|^2$, effectively defining a radius of influence for the characteristic point (see figure 5, left). Now $\widetilde{\Pi}_\phi$ is for each shape $\phi \in \Phi$ given as

$$\widetilde{\Pi}_\phi(\mathbf{p}) = g_{\sigma_\phi}(\|\Lambda_\mathbf{p} - \Lambda_\phi\|)$$

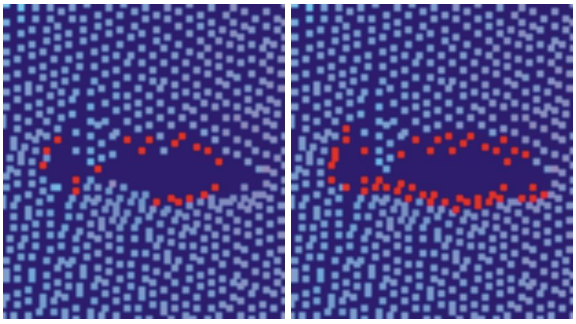
Obviously, the regions for different shapes overlap. Therefore we normalise and define

$$\Pi_\phi(\mathbf{p}) = \frac{\widetilde{\Pi}_\phi(\mathbf{p})}{\sum_{\varphi\in\Phi}\widetilde{\Pi}_\varphi(\mathbf{p})}.$$

4.5 Combining the Criteria

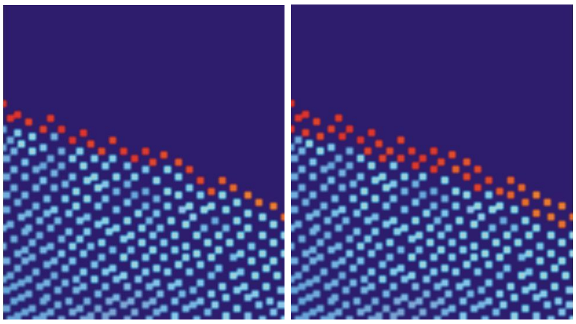
Every criterion has its own advantages. Compared to the angle criterion, the halfdisc criterion is better capable of detecting small holes, especially when the hole is crossed by some edges of the neighbourhood graph, see figure 6.

On the other hand, while the halfdisc and the ellipsoid criterion typically find narrow bands of boundary



(a) (b)

Figure 6: A small hole, crossed by some edges of the neighbourhood graph G. Points with a boundary probability (as computed with the angle criterion (a) and the halfdisc criterion (b)) above a threshold of 0.5 are coloured in red.



(a) (b)

Figure 7: Boundary points detected by the halfdisc criterion form a band of boundary points (b), whereas the angle criterion finds a sharp boundary (a).

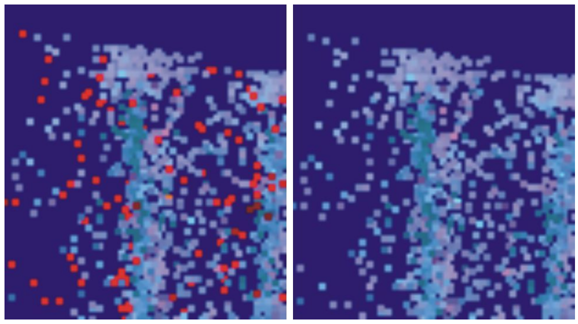
points around holes (in particular for larger k) the angle criterion is sharper and better exposes thin lines of boundary points (see figure 7). In the presence of noise, finally, the shape criterion performs best (see figure 8).
In order to make use of the different capabilities of the criteria and to increase the robustness of the boundary probability computation, we derive a combined boundary probability into a weighted sum

$$\Pi(\mathbf{p}) = w_\angle \Pi_\angle(\mathbf{p}) + w_\mu \Pi_\mu(\mathbf{p}) + w_\phi \Pi_{\text{Boundary}}(\mathbf{p}).$$

The weights $w_\angle$, w_μ and w_ϕ, where $w_\angle + w_\mu + w_\phi = 1$, can be adjusted by the user upon visual inspection. As default, a uniform weighting scheme produces good results, but for noisy models, the weight of the shape criterion should be increased.

4.6 Normal Estimation

Both, the angle and the average criterion, depend heavily on the normal at the point p. Therefore, if the data comes without normal information, a good estimation



(a) (b)

Figure 8: The angle criterion (a) identifies many false boundary points in the presence of noise, while the shape criterion (b) is not affected.

of the normal is mandatory. Following the normal estimation method by Hoppe et al. [HDD+92], the normal is given as the eigenvector corresponding to the smallest eigenvalue of the weighted covariance matrix of $N_{\mathbf{p}}$. In addition to that, we integrate information gath-

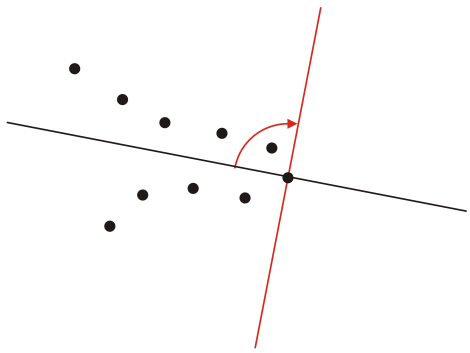


Figure 9: In sharp creases the fitting plane sometimes becomes normal to the surface. These cases can be detected with the angle criterion and the normal can then be flipped.

ered during the computation of the angle criterion in the normal estimation process as suggested in [LP02]. Sometimes, at sharp creases, the fitting plane is normal to the surface, see figure 9. To detect this situation, the angle criterion is evaluated after the normal has been estimated. If the boundary probability $\Pi_\angle(\mathbf{p})$ exceeds a given threshold, the estimated normal is rotated by 90 degrees about the axis defined by the two points on both sides of the maximum gap, projected into the tangent plane. Then the angle criterion is evaluated again, this time using the rotated normal, and the new normal is kept if the boundary probability has been reduced significantly, i.e. by more than 50%. This helps at sharp creases where sometimes the fitting plane is normal to the surface, see figure 9.
Please note that the estimation algorithm does not yield consistently oriented normals. Although this is required for neither of our criteria, it can easily be

achieved by applying the minimum spanning tree traversal introduced in [HDD+92] on the neighbourhood graph. We use this approach for visualisation purposes.

5 Boundary Loops

The extraction stage of the boundary detection algorithm aims at producing a classification for each point, stating if it is a boundary or an interior point. Here, in addition to the boundary probability computed with the scheme described in the last section, we will exploit the coherence between boundary points. This greatly improves the robustness of our method. Moreover, connected loops of points, circumscribing a hole, will be found, providing immediate access to the boundary.

5.1 Boundary Coherence

Any point on a boundary loop has at least one neighbour to each side also belonging to the boundary. This property can easily be exploited using a simple iterative classification step. First, all points with a boundary probability greater than a user defined threshold are declared boundary points. Then, for each of these points, the two neighbours forming the maximum gap in the sense of the angle criterion are found. A point stays classified as boundary point if and only if both of these neighbouring points have also been declared boundary points. All other points are marked as interior points. This process is repeated until no more points change their status. Note that only the neighbours of points that did change status in the previous iteration have to be reconsidered in the following step, making the classification very efficient.

After the classification, we use an algorithm that is built upon the one presented in [GWM01] to construct a minimum spanning graph (MSG) based on the proximity graph G. By construction, this MSG will contain loops if and only if they correspond to the boundary loops we are interested in.

To this end, we use an extension of Kruskal's minimum spanning tree algorithm. The required *edge weights* $w(i,j)$, are derived similar to [GWM01] in two parts. The first component penalises the boundary probability of the adjacent points:

$$w_{\text{prob}}(i,j) = 2 - \Pi(\mathbf{p}_i) - \Pi(\mathbf{p}_j).$$

The second component incorporates the local sampling density measured by $r_{\mathbf{p}}$ (defined as the average distance to $\mathbf{p}$'s neighbours (see sec. 4.3) and penalises long edges so that the boundary loops will contain as many boundary points as possible:

$$w_{\text{density}}(i,j) = \frac{2\|\mathbf{p}_i - \mathbf{p}_j\|}{r_{\mathbf{p}_i} + r_{\mathbf{p}_j}}.$$

The total weight is then given by

$$w_{\text{total}}(i,j) = w_{\text{prob}}(i,j) + w_{\text{density}}(i,j).$$

The construction of the MSG uses an extension to Kruskal's minimum spanning tree algorithm. In the beginning, every vertex of G comprises a stand-alone component in G. Then all eligible edges are processed in ascending order, according to their weight. Here, an edge (i,j) is considered eligible only if $w_{\text{prob}}(i,j)$ and $w_{\text{total}}(i,j)$ are below pre-defined thresholds. A threshold combination of 1.1 and 3 proved good in our experiments and was used for all the examples given in this paper.

If an edge (i,j) connects two distinct components of G, the edge is added to the MSG and the two components are joined. If, on the other hand, the edge connects two vertices of the same component, it is included in the MSG only if the emerging loop is longer than a predefined minimum loop length e, measured as the number of edges in the loop. Similar to the radius ϵ in the construction of the neighbourhood graph G, the minimum loop length e steers the minimal hole radius to be found. Therefore, we link these two parameters and set $e = \frac{2\pi\epsilon}{d}$, where d is the average edge length in the graph.

5.2 Loop Extraction

With the MSG at hand, the boundary loops can be extracted using a breadth first search. The search is started once for each vertex in the MSG, unless it has become part of a loop already. The algorithm maintains for all vertices a colour value signaling one of three states: white (untouched vertices), grey (queued for visitation) or black (already processed). In the beginning, all vertices are white, except the origin, which is grey (see figure 10). In every step the vertex on the front is marked black, removed from the queue of grey vertices, and all its white adjacent vertices are marked grey and appended to the queue. If an adjacent vertex is black, it is ignored, but if one of the adjacent vertices encountered is grey, a loop has been found and can be extracted by tracing back the steps of the breadth first search. In a final step, points belonging to a loop are marked as boundary points. The process is illustrated in figure 10.

6 Results and Conclusions

We applied our algorithm to a variety of models. Figure 11 illustrates the effect of our hole detection method using the symmetric neighbourhood graph that is designed particularly to filter out even abrupt sampling density changes, a situation which causes even well-established hole criteria to fail. For this example,

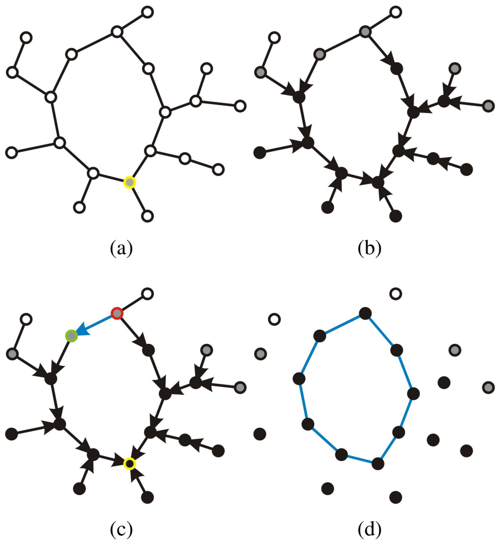


Figure 10: The extraction of a loop in one component of the MSG. (a) The breadth first graph traversal is spawned at the highlighted vertex (grey in the beginning). (b) The state of the vertices after four steps of the search. All grey vertices are queued for visitation, black vertices have been visited. Arrows indicate the vertices' predecessors. (c) When the adjacent vertices of the green vertex are examined, the grey vertex (red) is encountered and a loop has been found. The loop is extracted by tracing back the predecessors of both vertices. (d) The extracted loop.

one half of the depicted data set was heavily downsampled and only the angle criterion employed. Note how well the drastic change in sampling density is handled. Although this novel neighbourhood construction already considerably improves the performance of the so-called angle-criterion, the robust detection of holes in the presence of noise or also of holes of small size remains a challenge using only this criterion. To overcome this, we presented two novel boundary criteria: The halfdisc criterion is the 3d-analogue to the well-known border detection in images, whereas the ellipsoid criterion exploits a classification scheme based on local data analysis.

The notion of a hole is inherently and per-se ill-defined in the context of point set surfaces, and hence any classification ultimately needs to adapt to the application's (or rather the user's) interpretation. Consequently, our probabilistic approach can be trimmed using intuitive parameters, rendering the method easily adjustable to the task at hand. The parameter k of the neighbourhood definition determines the size of the local neighbourhoods. If k is increased, only larger holes can be detected, as smaller holes will be crossed by edges of G. We typically used a value between 12 and 25 for our test cases, depending on the amount of noise

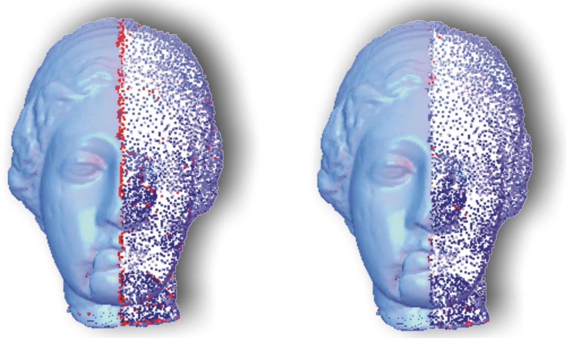


Figure 11: The effect of the symmetric neighbourhood relation. Left: k-nearest neighbours Right: Symmetric neighbourhood graph

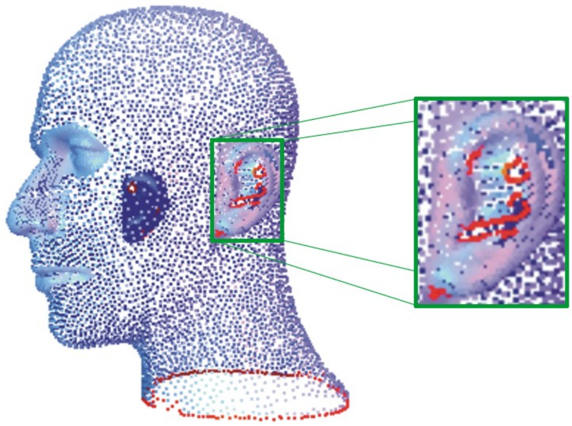


Figure 12: Boundary points identified in the mannequin model (points only) with $k = 15$. The top right image is taken from [DG01].

present in the data. If there is considerable noise, larger values of k can be used to improve the robustness of the hole detection, while the parameter ϵ can be used to define a minimum hole size, since the neighbourhood will stretch over all holes with a diameter less than ϵ. This way the user is enabled to focus on the important holes in the dragon for instance, as demonstrated in figure 14.

By making use of the coherence between boundary samples, the robustness of the hole detection is further increased. As a by-product of this stage, boundary loops are extracted, delivering subsequent processes direct access to the contours of the holes.

For many applications, such as automatic hole filling, the detection of holes has to be repeated after filling part of the hole. A reasonable efficiency of the hole detection is therefore desirable. In the dragon example (containing over 400000 points) the holes depicted in figure 14 (right) were detected in less than two minutes on a AMD Athlon 2.21 GHz processor. Specifically, the timings were: Construction of the kd-tree and the symmetrised proximity graph $23s$, computation of the

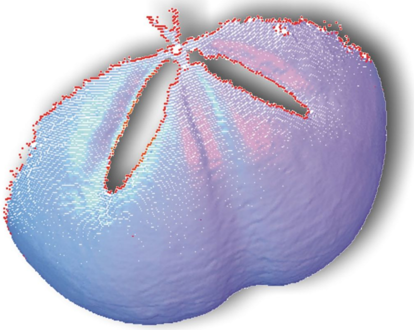


Figure 13: Boundary found in a scan of an echinite. All three criteria were combined with equal weight.

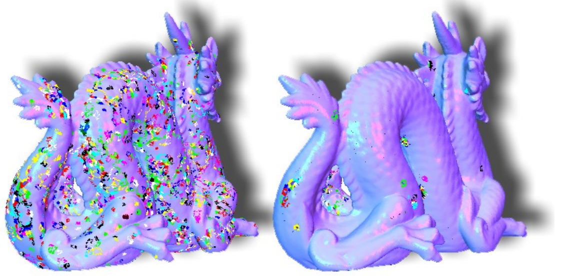


Figure 14: Numerous small holes are detected in the dragon model for $k = 15$, but larger holes can be isolated if all points within 0.01 of the bounding box diagonal are also included in the neighbourhood.

integrated boundary probability $46s$, extraction of the boundary loops $36s$. In the context of hole filling, the update of the boundary loops can naturally be performed incrementally, such that here timings can be expected to be even considerably faster; this has not been in the scope of this paper, though.
Figure 12 shows that our method extracts holes in the mannequin point cloud comparable to those identified for the corresponding mesh in [DG01]. Here, a classification step with a threshold of .3 was applied. In figure 15 the boundary of a single scan of the bunny has been extracted as a loop. A minimum loop size of $e = 1000$ was used to suppress the detection of loops around the smaller holes.

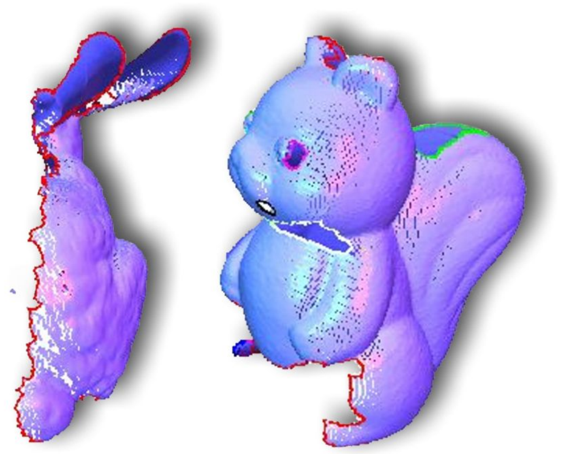


Figure 15: Detected boundaries in a single scan of the bunny and in the registered, yet incomplete squirrel data set.

References

[ABE98] N. Amenta, M. Bern, and D. Eppstein. The crust and the -skeleton: Combinatorial curve reconstruction. *Graphical models and image processing: GMIP*, 60(2):125–135, 1998.

[BK05] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611 – 621, 2005.

[BSK05] Gerhard Heinrich Bendels, Ruwen Schnabel, and Reinhard Klein. Fragment-based surface inpainting. Poster proceedings of the Eurographics Symposium on Geometry Processing 2005, July 2005.

[CN04] Moenning C. and Dodgson N.A. Intrinsic point cloud simplification. In *Proc. 14th GrahiCon*, volume 14, Moscow, September 2004.

[DG01] Tamal K. Dey and Joachim Giesen. Detecting undersampling in surface reconstruction. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 257–263. ACM Press, 2001.

[GWM01] Stefan Gumhold, Xinlong Wang, and Rob McLeod. Feature extraction from point clouds. In *Proceedings of 10th International Meshing Roundtable*, pages 293–305, Sandia National Laboratories, Newport Beach, CA, October 2001.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78. ACM Press, 1992.

[LP02] Lars Linsen and Hartmut Prautzsch. Fan clouds - an alternative to meshes. In T. Alano, R. Klette, and Ch. Ronse, editors, *Proceedings Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology and Computational Imaging*, page [10]. Springer-Verlag Berlin Heidelberg, 2002.

[PKKG03] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.

[SACO04] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Trans. Graph.*, 23(3):878–887, 2004.

Reusing frames in camera animation

Àlex Méndez-Feliu
Universitat de Girona
Institut d'Informàtica i Aplicacions
Edifici P4, Campus de Montilivi
17071, Girona, Spain
amendez@ima.udg.es

Mateu Sbert
Universitat de Girona
Institut d'Informàtica i Aplicacions
Edifici P4, Campus de Montilivi
17071, Girona, Spain
mateu@ima.udg.es

László Szirmay-Kalos
Technical University of Budapest
Informatics Building, B320
1117 Pázmány P. sétány 1/D.
Budapest, Hungary
szirmay@iit.bme.hu

ABSTRACT

Rendering an animation in a global illumination framework is a very costly process. Each frame has to be computed with high accuracy to avoid both noise in a single frame and flickering from frame to frame. Recently an efficient solution has been presented for camera animation, which reused the results computed in a frame for other frames via reprojection of the first hits of primary rays. This solution, however, is biased since it does not take into account the different probability densities that generated the different contributions to a pixel. In this paper we present a correct, unbiased solution for frame reuse. We show how the different contributions can be combined into an unbiased solution using multiple importance sampling. The validity of our solution is tested with an animation using path-tracing technique, and the results are compared with both the classic independent approach and the previous unweighted, biased, solution.

Keywords: Animation, Ray tracing, Path reuse, Global illumination, Path tracing

1 INTRODUCTION

In global illumination an image can be computed by tracing paths from the eye (or observer position) trough the pixels that compose the image plane towards the surfaces of the scene. In the path-tracing technique, from the hit point in the scene a random walk is followed, gathering the energy at every new hit point. The main drawback of these Monte Carlo random walks is the high number of paths needed to obtain an acceptable result. To obtain an animation or a sequence of frames in a global illumination framework with production quality, we need to cast many rays per pixel. Each frame has to have high accuracy to avoid both noise in the frame and flickering from frame to frame. An efficient solution to reduce this cost has been presented for camera animation [9]. Computation for one frame is reused for other frames via reprojection of the first hit of the eye ray to neighbour eyes positions. Although very computationally efficient, this solution is biased, as it does not take into account the different probability densities that generated the different contributions to a pixel. In this paper we improve on this solution by presenting an unbiased method for frame reuse. Our approach is a follow-up of previous research on path reuse [2, 17, 16]. We show how the different contributions can be combined to an unbiased solution using multiple importance sampling [20]. We test our solution with an animation using the path-tracing algorithm for global illumination, and compare it with a classic independent solution and the previous unweighted, biased, technique. Although the results are shown in this paper for the path-tracing algorithm, the validity of our technique is general.

This paper is organized as follows. In the next section we will refer to previous work on reusing frames in animation and on reusing paths in the context of radiosity and global illumination. Path-tracing is also shortly revisited. The theoretical foundation and algorithms of our techniques for reusing frames are presented in section 3. In section 4 details about the implementation and costs of our algorithm are explained. In section 5 results are presented that illustrate the benefits of our technique, and in the last section we present the conclusions and future work.

2 PREVIOUS WORK

2.1 Path-tracing algorithm

Random walks are a Monte Carlo common tool to solve second kind Fredholm integral equation [15, 7, 12]. For instance, they are used in global illumination and radiosity [1, 5] to solve the *rendering equation* [11]. Path-tracing [11], distributed ray-tracing [4], bidirectional path-tracing [13, 20], photon map [10] and Metropolis [22] are main global illumination techniques using random walk. In the path-tracing technique (see Fig.1a) an image is computed by tracing paths from the eye (or observer position) trough the pixels that compose the image plane towards the surfaces of the scene. To avoid a huge variance, from each hit point ray(s) are directed towards the light sources to gather light. The random walk can be terminated upon different termination conditions. Russian roulette is the most used termination technique. At a hit point, we decide at random whether to terminate or follow the path. Another possibility is to accumulate the albedos of visited hit points and stop when the accumulated albedo is below a given threshold. Path-tracing is a general, unbiased simple technique to compute global illumination, relatively easily to implement and very much appropriate to test improvements in global illumination, although more sofisticated techniques have been introduced like Metropolis and bidirectional path-tracing. On the other hand, shooting random walk solves the adjoint equation by simulating the trajectories of photons from the light sources, and hybrid methods like bidirectional path-tracing combine both shooting and gathering. The cost of a random walk simulation is mainly the cost of computing the next hit point, that is, the point visible from the old hit point in the sampled direction.

The main drawback of these Monte Carlo random walks is however the high number of paths needed to obtain an acceptable result. As the variance of the estimators is proportional to N^{-1} with N the number of paths, it makes necessary the use of many paths, of the order of millions, to obtain an acceptable noiseless image. This is still more dramatic in an animation computation, due to the high number of frames to be computed. Thus achieving some sort of path reusing can reduce the computational cost.

2.2 Reusing paths

Bidirectional path-tracing [13, 20] can be considered as the first attempt to reduce the cost by reusing paths. This method joins sub-paths belonging to the same pixel or to the same source point. However, the idea of the reuse of full paths and for different states (i.e., pixels, patches or light sources) was first presented by John Halton in [6] in the context of the random walk solution of an equation system. This technique was applied by Bekaert et al. in the context of path-tracing in [2, 16] (see Fig.1a), combined with multiple importance sampling [21, 20] to avoid bias. Pixels were grouped in tiles, and paths belonging to a pixel in the tile were reused for the other pixels in the tile from the second hit point of the path. A speed-up of one order of magnitude was reported for fairly complex scenes. Havran et al. [9] presented the reuse of paths in a walk-through, that is, when the observer changes position. Paths cast from one observer position were reused for other neighbor positions. Their technique admitted motion blur and they applied it in the context of bidirectional path-tracing. Although obtaining a high speed-up, the method remained biased as the samples were not weighted with the respective probability. In the radiosity context Besuievsky [3] used the same set of lines to expand direct illumination from different light source positions. The source positions were packed in a bounding box and lines crossing this box expanded the power of all intersected positions. The drawback of this method is that lines are wasted if the source positions are not tightly packed. Moreover, this method is only valid for diffuse sources. Recently path-reuse has been applied to light source animation in [17, 18, 14].

3 FRAME REUSE

In this section the theoretical framework of our algorithm is introduced. We first introduce the basic *native* estimators, then we show how we can estimate the radiance from a different eye and finally how the radiance estimators obtained with paths from different eyes can be combined by multiple importance sampling.

3.1 Estimating the *native* radiance

In global illumination we are interested in the integrals $L^o(i) = \int_{A_i} L(p)dp$ where A_i is the area of pixel i, and $L(p)$ is the radiance from visible scene point x that reaches the observer at point o through point p in pixel i . Introducing a change of integration variable [19] we obtain

$$L^o(i) = \int_S L(x \to o)G(o,x)V_i(x)\frac{\cos^3 \theta_i}{f^2}dx, \qquad (1)$$

where the integration extends over all scene surface points S, θ_i is the angle between the normal of the screen plane and direction $\omega(x \to o)$ at the center of the pixel i, and f is the focal distance, i.e. the distance from o to the plane of the screen. $V_i(x)$ takes the value of 1 if x is visible through the pixel i and 0 otherwise. The geometric factor $G(o,x)$ is defined as

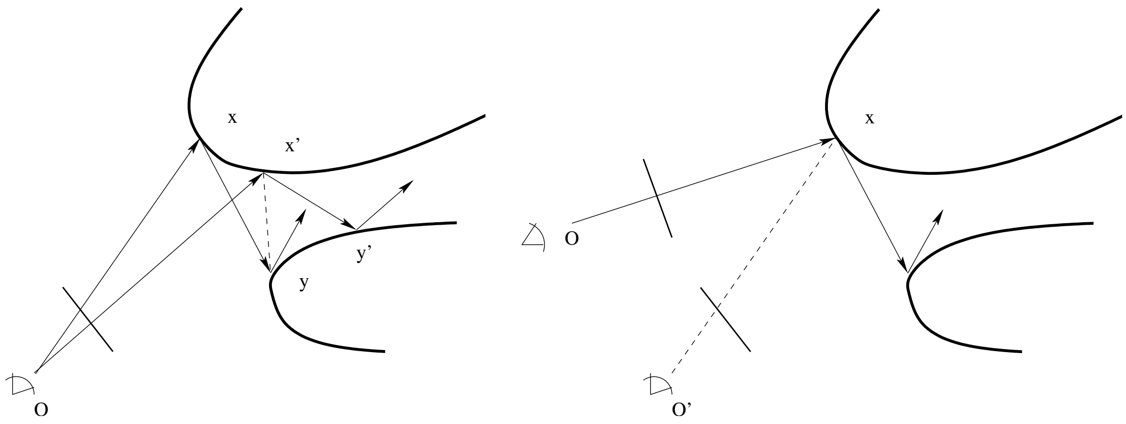
$$G(o,x) = vis(o,x)\frac{\cos(N_x, \omega(x \to o))}{d^2(x,o)} \qquad (2)$$

where $vis(o,x)$ is 1 if x and o see each other and 0 otherwise, N_x is the normal at point x, $\omega(x \to o)$ is the direction from x to o, and $d(x,o)$ is their distance. The radiance $L(x \to o)$ comes from the global illumination equation [11]:

$$\begin{aligned} L(x \to o) = L^e(x \to o) + \\ \int_\Omega \rho(\omega^{in}, x, x \to o) L(x, \omega^{in}) \cos(N_x, \omega^{in}) d\omega^{in} \end{aligned} \qquad (3)$$

where $L^e(x \to o)$ is the self emitted radiance, $\rho(\omega^{in}, x, x \to o)$ is the bidirectional reflectance distribution (brdf) function at point x, incoming direction ω^{in} [1] and outgoing direction $x \to o$. $L(x, \omega^{in})$ is the incoming radiance to x in direction ω^{in}.

[1] In fact, the true incoming direction is $-\omega^{in}$, but we use the opposite one to keep the reciprocity in the brdf.



(a) (b)

Figure 1: (a) Reusing a path from the second hit point, y, for a single observer O, creating thus the new path $O,x',y,\ldots$ at the cost of the visibility test $vis(x',y)$. (b) Reusing the path from observer O for observer O', at the cost of the visibility test $vis(O',x)$.

Substituting (3) into (1), and dropping constant terms and self-emission [2], we obtain

$$L^o(i) = \int_S \int_\Omega G(o,x)V_i(x)\rho(\omega^{in},x,x\to o) L(x,\omega^{in})\cos(N_x,\omega^{in})d\omega^{in}dx \qquad (4)$$

Primary estimator $\widehat{L^o(i)}$ for $L^o(i)$ is obtained by selecting a point x with probability $p_i^o(x)$ and then direction ω^{in} with probability $p(\omega^{in};x,x\to o)$. An unbiased estimator for $L(x,\omega^{in})$, $\widehat{L(x,\omega^{in})}$, can be obtained by any suitable technique, for instance by the random walk path-tracing technique. Thus

$$\widehat{L^o(i)} = \frac{G(o,x)V_i(x)\rho(\omega^{in},x,x\to o)\widehat{L(x,\omega^{in})}\cos(N_x,\omega^{in})}{p_i^o(x)p(\omega^{in};x,x\to o)} \qquad (5)$$

With importance sampling we select probabilities

$$p_i^o(x) \propto G(o,x)V_i(x)$$

and

$$p(\omega^{in};x,x\to o) \propto \rho(\omega^{in},x,x\to o)\cos(N_x,\omega^{in})$$

Inthis case the estimator becomes:

$$\widehat{L^o(i)} = a(x,x\to o)\Omega_i \widehat{L(x,\omega^{in})} \qquad (6)$$

where Ω_i (solid angle subtended by pixel i) and $a(x,x\to o)$ (albedo) are the probabilities normalization constants. Estimator (5) is the unbiased *native* estimator for a pixel, that is, the one obtained by sending rays from the observer through the pixel.

[2] Self emission can be easily dealt with separately.

3.2 Estimating the radiance from a different eye

Consider now a different observer o' (see Fig.1b). This observer will see x through a different pixel, j. We can obtain a (biased) estimator for $L^{o'}(j)$ reusing the value obtained for the radiance at x with estimator $\widehat{L(x,\omega^{in})}$ (this comes to reusing the path from x supposing the estimator is a random walk, see Fig. 1b). The estimator for pixel j from eye o' obtained with a ray started from eye o is given by the following expression:

$$\widehat{L^{o',i}(j)} = \frac{G(o',x)V_j(x)\rho(\omega^{in},x,x\to o')\widehat{L(x,\omega^{in})}\cos(N_x,\omega^{in})}{p_i^o(x)p(\omega^{in};x,x\to o)} \qquad (7)$$

Note that the probabilities used in the denominator are the native probabilities used to find point x from eye o through pixel i, but they should be normalized with respect to pixel j as seen from eye o'. We have then to normalize $p_i^o(x)$ with respect to the new eye. Let us drop the assumption that probability $p_i^o(x)$ depends on pixel i as seen from o and through which the ray was generated (this is an approximation, considering pixels on a spherical screen). The corresponding normalization condition should be fulfilled

$$\int_S p^o(x)V_j(x)dx = 1 \qquad (8)$$

where $V_j(x) = 1$ if point x is visible from o' trough pixel j and 0 otherwise. Suppose now that we distribute eye rays from o with probability proportional to $G(o,x)$. To obtain probabilities $p^o(x)$ we have to find the normalization constant given by the integral:

$$I = \int_S G(o,x)V_j(x)dx \qquad (9)$$

Integral (9) can be interpreted as the solid angle from o that *sees* the portion of the scene seen from the solid angle subtended by pixel j from eye o' (Ω_j), see fig.

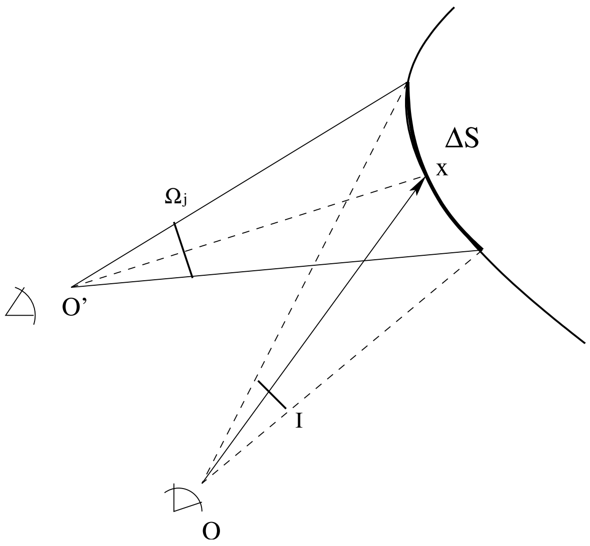


Figure 2: Here is shown in a graphical way the interpretation of equation (9). Ω_j is the solid angle subtended by pixel j, and I is the solid angle through which eye o sees what eye o' can see through Ω_j.

2. Observe that when $o = o'$ integral (9) is equal to Ω_j. Integral (9) is not known and computing it (using Monte Carlo integration) would require sending a lot of rays from o and comparing the visible or unoccluded hit points from o' to the total unoccluded+occluded. Lacking this information, we make the assumption that we have no visibility problems. Thus, given hit x from o obtained with probabilities proportional to $G(o,x)$, and taking into account that without occlusions $\Delta\Omega_j = G(o',x)\Delta S$ and $\Delta I = G(o,x)\Delta S$, the normalization constant (9) is approximated by

$$I \approx \frac{\Omega_j G(o,x)}{G(o',x)} \tag{10}$$

Expression (10) is used to normalize p_i^o and estimator (7) (having dropped the dependence on the pixel i) thus becomes

$$\begin{aligned}\widehat{L^{o'}(j)} &= \frac{G(o',x)\rho(\omega^{in},x,x\to o')\widehat{L(x,\omega^{in})}\cos(N_x,\omega^{in})}{\frac{G(o',x)}{\Omega_j G(o,x)}G(o,x)p(\omega^{in};x,x\to o)} \\ &= \frac{\Omega_j\rho(\omega^{in},x,x\to o')\widehat{L(x,\omega^{in})}\cos(N_x,\omega^{in})}{p(\omega^{in};x,x\to o)}\end{aligned} \tag{11}$$

and for a diffuse hit point ρ (and thus neither p) depends on the incoming eye ray, thus it becomes simply

$$\widehat{L^{o'}(j)} = \Omega_j \widehat{L(x)} \tag{12}$$

where $\widehat{L(x)}$ is the estimated radiance from hit point x.

3.3 Combining paths

In [9] an unweighted combination of estimators of kind (5) and (7) was done, resulting in a biased estimator. We present now a strategy that gives an unbiased estimator. For each pixel and frame, we keep accumulated radiance value and native ray estimators (among many other data useful for our computation, see 4.3) generated with probability $p_i^o(x)p(\omega^{in};x,x\to o))$. When hits from neighbour frames can be reused for this pixel (suppose estimator $\widehat{L^{o,j}(i)}$, generated with probability $p_j^{o^j}(x^j)p(\omega^{in,j};x^j,x^j \to o^j)$), we combine them using multiple importance sampling with the native estimator. This gives the new unbiased estimator:

$$\widehat{L^o(i)} = \sum_j \frac{p_j^{o^j}(x^j)p(\omega^{in,j};x^j,x^j\to o^j)\widehat{L^{o,j}(i)}}{\sum_k p_k^{o^k}(x^j)p(\omega^{in,j};x^j,x^j\to o^k)} \tag{13}$$

We show now how this estimator is applied.

For the sake of simplicity, and without loss of generality, consider only two observers O_1 and O_2[3]. In this case estimator (13) becomes estimator (14) for observer O_1, using the approximation (12) for the estimator and also (9) for the normalization constant in the weights. We have taken the approximation that all pixels subtend the same solid angle. Remember also that the visibility boolean function is included in the G function. Consider first the particular case for diffuse hit pixels.

$$\begin{aligned}L(O_1) &= \frac{G(O_1,x_1)}{G(O_1,x_1)+G(O_2,x_1)\frac{G(O_1,x_1)}{G(O_2,x_1)}}L(x_1) \\ &+ \frac{G(O_2,x_2)}{G(O_1,x_2)\frac{G(O_2,x_2)}{G(O_1,x_2)}+G(O_2,x_2)}L(x_2) \\ &= \frac{1}{2}L(x_1)+\frac{1}{2}L(x_2)\end{aligned} \tag{14}$$

Thus approximation (10) for the normalization constant leads to the simple unweighted estimator when we deal only with diffuse hits, and this is why Havran et al. solution [9] works well for diffuse surfaces.

For the non-diffuse general case, using again approximation (10) allows us to eliminate all G terms, and using estimator form (11) we obtain

$$\begin{aligned}L(O_1) &= \frac{\rho(\omega^{in,1};x_1,x_1\to O1)L(x_1,\omega^{in,1})\cos(N_{x_1},\omega^{in,1})}{p(\omega^{in,1};x_1,x_1\to O1)+p(\omega^{in,1};x_1,x_1\to O2)} \\ &+ \frac{\rho(\omega^{in,2};x_2,x_2\to O1)L(x_2,\omega^{in,2})\cos(N_{x_2},\omega^{in,2})}{p(\omega^{in,2};x_2,x_2\to O1)+p(\omega^{in,2};x_2,x_2\to O2)}\end{aligned} \tag{15}$$

where $L(x_1,\omega^{in,1})$ and $L(x_2,\omega^{in,2})$ are the incoming radiances to x_1 from direction $\omega^{in,1}$ and to x_2 from direction $\omega^{in,2}$.

4 IMPLEMENTATION

4.1 Algorithm

Once we hit a point in the scene from the eye, we can reuse this information for all the other frames in our camera animation, but only if the point is visible from the other eyes. For this reason and also because of memory restrictions, we are only interested in reusing

[3] For a clearer explanation we also drop here the albedo and solid angle Ω

```
for i = firstFrame to lastFrame do
    currentEye = getEye(i)
    for all pixel in images[i] do
        hit=traceRay(currentEye, pixel)
        for j = firstFrame to lastFrame do
            reuseHitinImage(hit,images[j])
```

Algorithm 1: The algorithm for the *hit harvest* phase, considering only the group of neighbouring frames.

the hit with the closest neighbouring frames, as the probability of being visible is much higher.

We will consider two phases in the computation of our animation frames. The first one is the *hit harvest* (see algorithm 1), where we find the native hit points, compute the rest of the gathering path, connect every hit point with the other eyes to see if they can be reused, and finally, if it is the case, we add a pointer to it in the list of outer hits of the corresponding pixel. Meanwhile, additional probabilities and reflectances needed for the computation are kept in memory. The second phase is the *image computation* itself, in which we use all the stored information, including the native and outer hits for every pixel to compute the final pixel color.

As a few seconds animation involves hundreds of frames, it is not feasible to keep all them in memory at the same time. We have two possible strategies. The first one is to reuse a hit in the n previous and subsequent frames keeping in memory all information needed for the final computation, that will be done once we know we are not reusing more hits for that frame, that is, the $(actual-n)$th frame. But as we can see in equations (15) and (13), for every hit that will be used in a pixel computation, we need to use all probabilities in combination with all the eyes that have been used in the other hits for that pixel. This means keeping in memory also information for frames previous to the $(actual-n)$th, or recompute these probabilities every time we need them. This is a waste of time or a waste of memory.

The second strategy consists in considering a group of $2n+1$ neighbour frames and reuse every native hit in all the other frames in the group, no matter if it is the first one, the last one or the one in the middle. When we are done for the group, instead of moving to the next $2n+1$ frames, we can move just one frame, overlapping $2n$ frames, but without deleting the previous results for the frames that are still active. This previous results can be simply averaged with the new ones. This is the strategy we follow.

4.2 Cost analysis

Now we analyze the relative cost of the animations with and without reuse. Suppose we cast n_r rays per pixel and reuse n_f frames at once. The cost of tracing an eye ray is c_e, the cost of computing the illumination at the hit point in the scene is c_l, and the cost of a visibility test is c_v. In the case of no reuse, the cost of computing n_f frames is $n_f n'_r(c_e+c_l)$. In the reusing case the cost is $n_f n_r(c_e+c_l)+n_f(n_f-1)n_r c_v$, where the second term in the sum accounts for reusing all rays. In the optimal case a ray through a pixel would be equivalent to a native ray, and we compare thus the cost of two animations with equal number of rays per pixel, that is, $n'_r \simeq n_f n_r$. The relative cost for this case is:

$$\frac{n_f n_f n_r(c_e+c_l))}{n_f n_r(c_e+c_l)+n_f(n_f-1)n_r c_v} = \frac{n_f n_r(c_e+c_l)}{n_r(c_e+c_l)+(n_f-1)n_r c_v} = \frac{n_f(c_e+c_l)}{(c_e+c_l)+(n_f-1)c_v} \quad (16)$$

This last expression has the limiting value $\frac{(c_e+c_l)}{c_v}$ when n_f tends to infinite. Supposing $c_l \gg c_e$, limit efficiency will be $\frac{c_l}{c_v}$.

Observe that the above limit efficiency is an upper bound, as on the one hand not all rays will be able to be reused, and on the other hand the variance associated with a reusing frames estimator is higher than with an independent one, because in the independent estimator we have the benefit of importance sampling.

A second, and very important, independent increase in efficiency comes from the reduction in flickering from frame to frame. This reduction is due to that reuse of paths for different frames correlates the computations for all them. And in the way we have constructed our algorithm there is no flickering shown when passing from a group of reused frames to the following one, as we interleave them.

4.3 Memory use

We need a huge amount of memory to keep track of all our computation. We have to keep not only all the images for the current group being computed (including native reflectances and hits), but also the lists of outer hits for every pixel and all possible combinations of probabilities and reflectances per pixel and frame.

Same as before, suppose we use n_h hits per pixel and reuse n_f frames at once. Images are made of $w \times h$ pixels, so we have a total number of pixels $n_{pix} = w \times h \times n_f$.

For every pixel we keep the final color and the number of samples to add and average every result. We also need the list of outer hits for every pixel. The total nodes of outer hits are $n_{nod} = n_h \times n_{pix} \times (n_f-1)$ and are distributed in lists among all the pixels. Every node contains four integers: the image number, the pixel (w and h) and the number of sample, thus it can point towards the data we need to reuse from another image. For every native hit in a pixel we have to keep

 ISBN 1213-6972 ISBN 80-86943-09-7

the native direct and indirect gathered radiance, cosinus weighted, and a n_f-vector containing all probabilities and reflectances if we combined the hit with the rest of eyes.

Just to see the numbers in a concrete example, if we are using 2 samples for every one of the 800×600 pixels and reusing groups of 7 images, we get 3360000 pixels ($800 \times 600 \times 7$), each containing a final color (3 floats), the number of samples (one integer) and a pointer to the list of nodes. We have 40320000 nodes ($2 \times 800 \times 600 \times 7 \times 6$), four integers and a pointer each. We also have the 6720000 native radiances to reuse (direct and indirect, 3 floats each) and 47040000 ($2 \times 800 \times 600 \times 7^2$) probabilities (1 float) and reflectances (3 floats). If we assume each float, integer and pointer is 4 bytes, we need a total memory of 1787520000 bytes for this structure. That's almost all the memory of our 2Gb pentium 4.

5 RESULTS

We have applied the proposed algorithm to an animation computed using path tracing. The frame resolution is 800×600 pixels. We rendered 48 frames, for a 2 seconds animation[4]. For every pixel, we used 2 samples and reused them in groups of seven neighbour frames. As these groups are overlapped, we get a maximum average in number of samples of 98 (2×7^2). The actual average is less than that (between 85 and 90) because some reuses are lost (they can lie out of frame or be hidden by other objects) and it mainly depends on distances between different neighbour eyes, i.e. the smoothness of the camera movement. If camera movement is not smooth or the number of neighbour frames to reuse is too high this ratio decreases, and noticeable differences of noise between different parts of the same image might appear. In our example, as our camera movement corresponds to a zooming of a glossy phong brdf vase, the pixels in the center of the images get more samples than those lying near the borders. This can be interpreted as an advantage, as perception focuses in the center of the image when zooming, some kind of perception driven sampling is performed. The first frames and the last ones present more noise because they cannot be overlapped with previous (in the case of the first ones) or subsequent (for the last ones) groups of frames. Computing time was approximately 40 hours, for a PentiumIV with 2 Gigabytes of memory. That is 50 minutes per frame. A single path tracing image without reuse and with 98 samples per pixel takes more than 5 hours to compute. It is more than 6 times faster, and it can be even faster if we reuse more frames. If we compare this animation with the one computed with Havran et al. method [9], we can appreciate almost no differences. This is because we have reused very few frames and they are very close to each other.

We have computed a second animation with a higher number of reusing frames to prove more clearly the differences between the methods. In this case one hit is reused in 17 frames. We used 2 samples per hit, so we get a maximum average in number of samples of 578 (2×17^2). The actual average is about 500 due to loss of reuses. Due to memory restrictions, resolution is now reduced to 320×240 pixels. Computation time is about 16 hours. This is 20 minutes per frame, almost 8 times faster than the computation time of a single path tracing image with 500 samples per pixel (155 minutes). If we look at Havran et al. version of the same animation we clearly appreciate more noise in the vase in the form of glittering.

In Fig. 3 we show the same frame (the middle frame in our second animation) obtained with three different computations. In the first one (image a) an image with no reuse has been computed using 500 samples per pixel. It takes more than two and a half hours to compute. Image b) shows biased Havran et al. [9] version for reuse of frames. Time computation results in about 18 minutes per frame. Image c) is the result of our unbiased version. It takes 20 minutes to compute, a little more time than b), but we need much more memory. Both images b) and c) present more noise than image a) near the border due to loss of reuses. Image b) presents more noise than image c) in the vase and other glossy objects. Diffuse objects look the same in images b) and c).

In Fig. 4, details for the vase are shown. First image a) is computed with no reuse and 15 samples per pixel. Image b) is biased and computed with the Havran et al. [9] version and image c) is computed with our unbiased method. Both are computed with 2 samples per pixel and reuse of three fairly separated frames, i. e., a maximum average in number of samples of 18 (2×3^2). We clearly see much more noise in image b).

6 CONCLUSIONS AND FUTURE WORK

We have presented in this paper an efficient unbiased method to combine frames in camera animation. It consists in reusing the incoming radiance information of a hit point for the neighbouring frames of the animation. The different probabilities are taken into account and multiple importance sampling technique is used to correctly combine the different samples. Our method makes the difference when using non-diffuse materials, because the diffuse ones distribute reflected rays with equal probabilities in all directions, and when the separation between reusing frames increases. In diffuse cases or when reusing frames are very close, other biased methods can work fine. The main drawback of our method is the large amount of memory needed for the computation.

[4] Animations can be found in http://ima.udg.es/~amendez/TIC2001/gal_hitreuse.html.

The new method has been demonstrated with an animation of a camera in a scene that contains a vase with a glossy brdf, computing the global illumination using the path-tracing technique.

Future work will be addressed to increase the efficiency of our approach using coherence in visibility computation, by guessing on the one hand the visibility for one observer from the results for neighbour observers and on the other hand by using an acceleration schema similar to [8]. We will also try to combine both the benefits of this approach and the reuse of paths for light source animation [18]. Combination with an adaptive sampling technique, i.e., using more native samples for those pixels that come with not enough outer hit samples, because they are occluded by other objects or because they lie near the bounder of the image.

ACKNOWLEDGEMENTS

This project has been funded in part with Gametools project from the European VIth Framework, with grant number TIN2004-07451-C03-01 from the Spanish Government and with Hungarian-Spanish Joint Action number HH2004-0011.

REFERENCES

[1] Philippe Bekaert. *Hierarchical and Stochastic Algorithms for Radiosity*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, 1999.

[2] Philippe Bekaert, Mateu Sbert, and John Halton. Accelerating path tracing by re-using paths. In *Rendering Techniques 2002 (Proceedings of the Thirteenth Eurographics Workshop on Rendering)*, pages 125–134, June 2002.

[3] Gonzalo Besuievsky. *A Monte Carlo Approach for Animated Radiosity Environments*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 2001.

[4] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 137–145, July 1984.

[5] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. AK Peters Limited, 2003.

[6] John Halton. Sequential monte carlo techniques for the solution of linear systems. *Journal of Scientific Computing*, 9(2):213–257, 1994.

[7] J. Hammersley and D. Handscomb. *Monte Carlo Methods*. Chapman and Hall, London, 1979.

[8] Vlastimil Havran, Jiri Bittner, and Hans-Peter Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *Proceedings of the Spring Conference on Computer Graphics 2003 (SCCG 2003)*, April 2003.

[9] Vlastimil Havran, Cyrille Damez, Karol Myszkowski, and Hans-Peter Seidel. An efficient spatio-temporal architecture for animation rendering. In *Proceedings of Eurographics Symposium on Rendering 2003*, pages 106–117. ACM SIGGRAPH, June 2003.

[10] Henrik Wann Jensen. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30. Springer-Verlag/Wien, 1996.

[11] James T. Kajiya. The Rendering Equation. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20(4):143–150, August 1986.

[12] M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods*. John Wiley & Sons, 1986.

[13] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[14] Àlex Méndez-Feliu and Mateu Sbert. Combining light animation with obscurances for glossy environments. *Computer Animation and Virtual Worlds*, 15(3-4):463–470, july 2004.

[15] R.Y. Rubinstein. *Simulation and the Monte Carlo Method*. Wiley Series in Probabilities and Mathematical Statistics, 1981.

[16] Mateu Sbert, Philippe Bekaert, and John Halton. Reusing paths in radiosity and global illumination. *Monte Carlo Methods and Applications*, 10(3-4):575–586, 2004.

[17] Mateu Sbert, Francesc Castro, and John Halton. Reuse of paths in light source animation. In *Proceedings of Computer Graphics International 2004 (CGI '04)*, pages 532–535. IEEE Computer Society, June 2004.

[18] Mateu Sbert, Laszlo Szecsi, and Laszlo Szirmay-Kalos. Real-time light animation. *Computer Graphics Forum (Eurographics 2004 Proceedings)*, 23(3):291–299, September 2004.

[19] László Szirmay-Kalos, Mateu Sbert, Roel Martínez, and Robert F. Tobler. Incoming first-shot for non-diffuse global illumination. In *Spring Conference on Computer Graphics*, Budmerice, Slovakia, 2000. Available from http://www.fsz.bme.hu/˜szirmay/puba.htm.

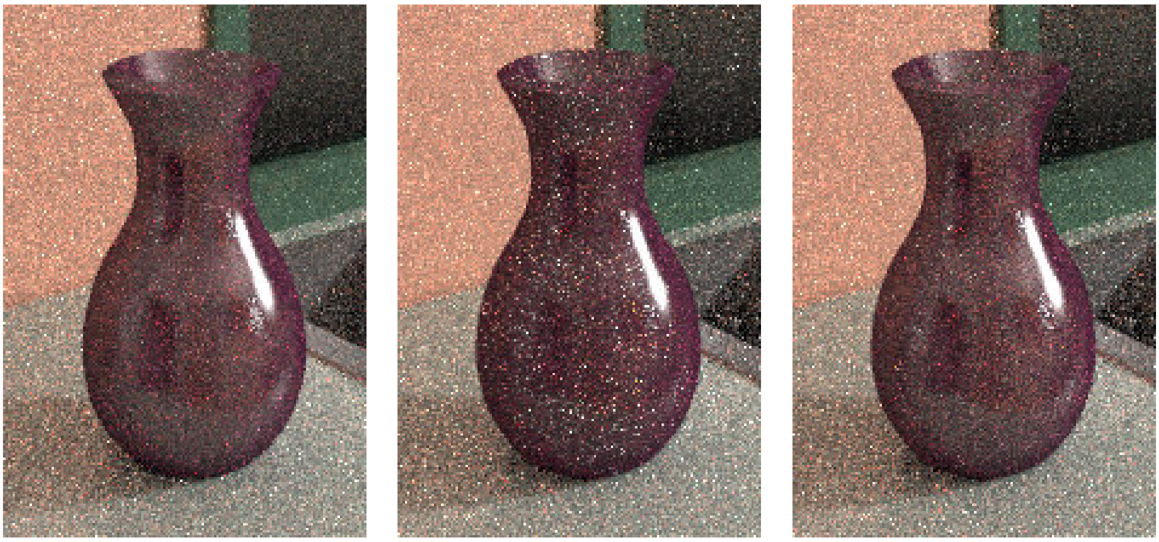
[20] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997. Available from http://graphics.stanford.edu/papers/veach_thesis.

[21] Eric Veach and Leonidas J. Guibas. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Computer Graphics Proceedings, Annual Conference Series, 1995 (ACM SIGGRAPH '95 Proceedings)*, pages 419–428, 1995.

[22] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, pages 65–76, 1997.

(a) (b) (c)

Figure 3: Here we show the same frame (the middle frame in our animation) obtained with three different computations. In the first one (image a) an image with no reuse has been computed. It takes more than 2,5 hours to be computed. Image b) shows Havran et al. [9] version for reuse of frames and takes 18 minutes to compute. Image c) is the result of our unbiased version and takes 20 minutes. The unbiased c) version uses much more memory. Images b) and c) presents noise near the border due to loss of reuses, and image b) presents noise in glossy objects due to biased computation.



(a) (b) (c)

Figure 4: The differences between the methods are clearly appreciated for non-diffuse materials when we reduce the computation time (noise is higher, consequently) and the separation between frames increases. Here we see the details of the vase for the 800×600 image when reusing only 3 frames fairly separated. First image a) is computed with no reuse. Image b) is biased and computed with Havran et al. [9] version and image c) is computed with our unbiased method. We clearly see much more noise in image b).

 ISBN 1213-6972 ISBN 80-86943-09-7

Similarity Brushing for Exploring Multidimensional Relations

Matej Novotný
Comenius University Bratislava
mnovotny@fmph.uniba.sk

Helwig Hauser
VRVis Research Center, Vienna
hauser@vrvis.at

ABSTRACT

Displaying multidimensional information has always been a challenge. Projecting multiple dimensions into a two dimensional display is one of the core tasks of information visualization. The human visual system is limited to a low number of dimensions and therefore the human-oriented projection does not easily combine the whole information contained in the original space. This paper introduces a new interaction tool, that implants the *n*-dimensional information into a low dimensional view and bridges the projection space with the original space in an intuitive and simple way. In one direction the tool performs *n*-dimensional data-driven brushing based on screen space interaction. In the opposite direction it allows for interactive visual exploration of the original multidimensional space in an infovis display. The implementation is presented using a standard scatterplot but it can be extended to many other infovis techniques as the concept does not depend on the screen space configuration.

Keywords: Information visualization, brushing, selection, multiple dimensions, interaction, scatterplot.

1 INTRODUCTION

analysis of multidimensional information is a widely spread and important task. Many domains generate and handle data of multiple attributes e.g. physical simulations, biochemical data or stock market information. The raw data themselves contain a lot of knowledge but almost none of it reveals without analysis. Many techniques were developed to support the knowledge discovery and two basic directions of research can be observed. One of them exploits the processing power of computers to work out the knowledge in an automatic way using statistical or data mining methods. The drawbacks of the automatic methods are usually lack of semantics or non-linear logic. Therefore the second approach takes advantage of abstract thinking and domain knowledge of human and often uses visualization-based interfaces to analyze the data. Both human and computer have their own qualities that predetermine them each for specific (and usually different) tasks.

In data analysis domain these two powerful processors are now being used in conjunction and the resulting techniques try to take the best of both worlds to complete their tasks. The power, storage and precise computations of a machine are being combined with the intuition, experience-based judgment and common or domain knowledge.

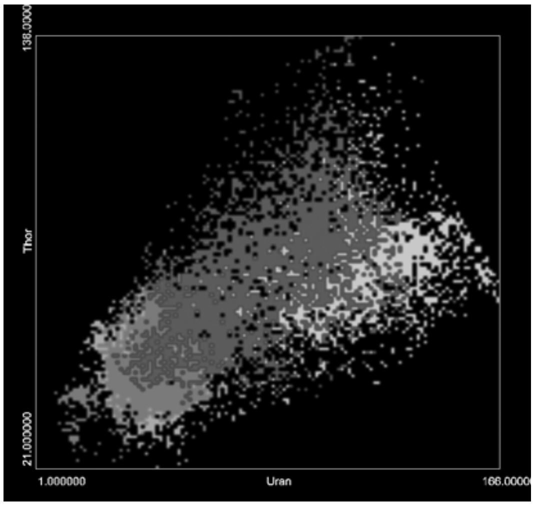


Figure 1: A complex *n*-dimensional segmentation performed in a simple scatterplot. These overlapping and fuzzy segments would require arduous effort to select if only standard brushing was applied. Please refer to [23] for full color figures.

One of the advantages of computers over humans is the ability to handle high-dimensional information. To compensate for this, the broadest information channel (the human visual system) is popularly used to communicate between computers and humans. Numerous visualization systems operate these days to support this type of information exchange. But when exploring multidimensional information through the means of computer visualization, one usually faces the problem of the low dimensional graphical interface between the hu-

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

man and the computer [20] Our solution presents a way to combine user-driven analysis with the power of automatic processing in order to effectively observe, explore and analyze multidimensional information in any common visualization technique (Figure 1).

1.1 Multidimensional visualization

Numerous solutions for visualization of multivariate data exist but, no matter how precise they are, eventually they bring up the question of how much does a low dimensional projection correspond to its original multidimensional source. The link between the 2D display and the original *n*D data leads through the projection. The action (be it either a selection or an observation) performed in the display extrapolates to the data space in order to match the original multidimensional context. Even though the re-projection extends the 2D action into *n*D, its nature remains two-dimensional. Another 2d action (usually using a different view) has to be presented to refine the action and such a refinement often has to be performed several times to satisfyingly approximate the desired *n*D action through a combination of multiple 2D actions [22], [2].

The tool presented in this paper, called the similarity brush, combines automatic and human-based processing in a way that overcomes the dimensionality bottleneck of a computer display. This is feasible using the presumption that the samples similar one to another are often parts of the same structure regardless of their dimensionality. This enables to bridge the screen space and the data space through similarity information that captures the *n*D structures inside the data. Thus high-dimensional relations can be explored by user in a single 2D display and the interaction with the 2D display connects directly to the original data space where automatic techniques take place. The similarity brush provides a new means to focus user attention and to steer the exploratory process inside a multidimensional environment.

The fact that various similarity measures used to abstract multidimensional information have been heavily investigated in the data mining society [15], [16] creates a reliable theoretical background for abstracting the *n*D information and makes the presented concept a promising framework for visual exploration of multidimensional data.

The tool and the idea behind it are further explained in Sections 3 and 4. Examples of using the similarity brush together with comments on them can be found in Section 5. The related work is addressed in Section 2.

2 RELATED WORK

The need for an accurate display of multivariate data is one of the most motivating stimuli for information visualization. The techniques that visualize multivariate information are basically twofold. Either they reduce the number of dimensions (by dimension subsetting or dimension reduction) so that intuitive visualization methods can be used or they display all dimensions using various sophisticated designs [9] (dimensional stacking, dimension embedding or axis reconfiguration.) Dimension reduction techniques such as principal component analysis [10], self organizing maps [11] or multidimensional scaling [13] produce a low-dimensional representation of the data while trying to preserve most of the multidimensional information. In our approach this information is condensed in a function that describes similarity between two data entries.

The similarity brush uses this function to produce a data-based selection that is derived from a user specified screen-based brush. The idea of data-driven brushes was successfully implemented in the structure based brushes [9] to perform selection in data space, but a hierarchical structure for the data had to be provided beforehand. An attempt to perform data-driven brushing was presented by Martin and Ward [14]. Their solution operates only on the two-dimensional data subspace identical to the screen space and the brush is eventually ruined by being transformed to a combination of regular one-dimensional value-based queries, which naturally includes many undesired entries into the result.

Our approach protects the multidimensional nature of a data-driven brush and works without any a priori given hierarchy. Moreover it stores separate information about the screen-based brush and the derived data-driven brush to enable further refinement in both the data space and the screen space. These two brushes are combined using a framework described in Section 3.2. The framework extends and formalizes various previous approaches to brush combination [5], [21] and balances the combination of two different information spaces.

2.1 Interaction

One way to deal with the limitations of an infovis display leads through changing the parameters of the visualization or performing user-driven operations on the data. Manipulation with the display is a crucial part of the visual exploration. Especially if the data are multivariate and the user has to change his focus, operate with different views, refine his actions or adjust the display to fit his/hers needs. As described in [7], through a realtime interaction with the display the user immerses himself in the data and if the connection between his actions and the reaction of the display is appropriate, even complex structures can be perceived in a 2D display [12].

An important part in interactive exploration is defining the area of interest. This paper addresses this problem by combining *n*D brushes and 2D interaction, which allows to perform multidimensional selection

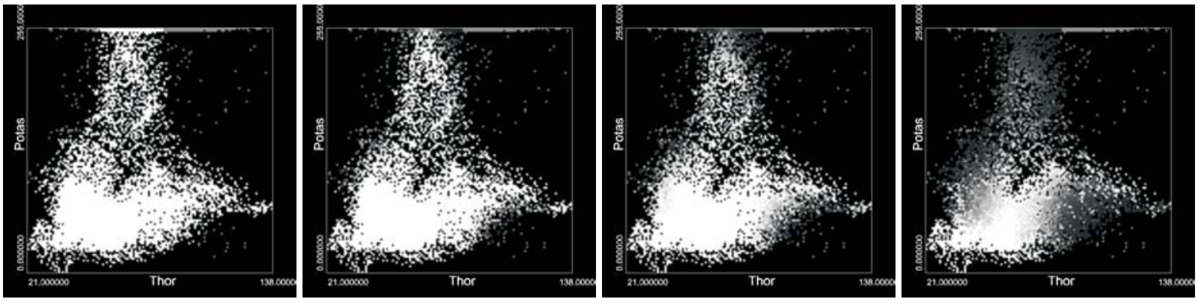


Figure 2: The basics of similarity brushing – primary selection inside the remote sensing data set is conducted on screen as a combination of different local brushes (marked yellow.) The data-driven selection (marked red) is derived from it by decreasing the similarity threshold.

operation using only standard interaction metaphors like brushing or dynamic queries [1], [24].

In a wide area of data analysis tasks the area of interest is not known beforehand and the exploratory process involves looking for interesting structures or patterns in the data. The hereby presented approach takes advantage of another well-known metaphor – the Magic Lens [4] – and uses it to integrate information of *n*D nature into the screen space. The examples demonstrating the advantages of these new interaction options can be found in Section 5.

2.2 Scatterplot

We chose Scatterplot to illustrate the benefits of the similarity brush, as it is a popular and very powerful visualization technique. Scatterplot [3] is plainly an orthographic projection of *n*-dimensional data space into a two-dimensional subspace determined by particular two of the original dimensions. The greatest advantage of the scatterplot lies in the ability to show two-dimensional relations in an instant thanks to the projection that preserves the basic spatial relations. The drawback of the simplicity of the scatterplot are the limitations of the displayed dimensions. Structures exceeding the two specified dimensions or those that are overplotted might get lost in the scatterplot. A different view is usually necessary to improve the visualization. The similarity brush overcomes this limitation and provides valuable information from "behind the scenes". With the help of the similarity brush many new structures that could not be seen before are revealed, mainly those of higher dimensionality or with unsharp and overlapping borders.

3 SIMILARITY BRUSHING

In this section we describe our new approach to jointly operate in visualization space as well as also in data space when interacting with the data, e.g., while selecting data subsets of special interest or during interactive data exploration. Below, we first describe the basic idea of *Similarity Brushing* before we go into details.

3.1 Similarity Brushing – The Basic Idea

For similarity brushing we consider a visualization scenario in which an *n*-dimensional dataset D (with n usually being around 5 to 50) is visualized in an m-dimensional visualization space V (with $m < n$ and m usually being 2 or 3), i.e., a scenario in which the visualization transform $\mathbf{p} : D \rightarrow V$ introduces a loss of dimensionality. As well known also from a lot of related work, it is difficult (or sometimes even impossible) to properly represent the *n*-dimensional relations between the data items of D in the *m*-dimensional visualization. Through $\mathbf{p}$, it is usually well possible that data items, which are far apart from each other in the *n*-space, lie near to each other in the *m*D visualization. Accordingly, it easily can happen that data substructures (like data clusters), which clearly are delimited in *n*-space, show up intermingled in the visualization and therefore cannot be visually differentiated apart from each other.

Similarity brushing now enables the user to jointly address structures in the visualization, i.e., data structures which are preserved by transform $\mathbf{p}$, as well as also structures, which only show up in the original *n*D data space. The basic idea of similarity brushing is as follows:

Working in visualization space: First, the user interactively marks a certain structure in the visualization (like in standard brushing) to select some data items for further investigation. The prime example here is that the user marks the core of a data structure of interest in the visualization. This can be one data point only, an entire subset of the data, or even a larger part of the visualized data items.

Working in data space: Next, the user extends this first brush to also include further data items which are similar to the already selected data items. The important thing here is that now a distance metric for the *n*D data space is used (instead of measuring distances in visualization space). Thereby, only those data items are added to the original brush which also are near to the previously selected ones in the *n*-space. To continue our prime example, the user would thus extend his/her first (quite conservative) selection to also include all other data items of the spotted data structure (but without touching all those data items which only seem to be part of the structure, but not really are – at least in terms of distances in *n*-space). An example is depicted in Figure 3.

The advantage of similarity brushing is that we can exploit the advantages of the visualization as well as of a data-centered approach (such as data mining): (1) The *m*D data visualization usually provides the user with a very intuitive interface to the data – the user literally sees the data in front of him/her. The human visual system is very powerful in detecting interesting structures/subsets in such a visualization. Accordingly, it is very useful to allow the visualization-based selection of data items (as long used under the term brushing). (2) Our approach to extend (not substitute) this interaction also to data space allows to overcome situations where disadvantages of the visualization become apparent such as the loss of dimensionality that leads to ambiguities in the visualization.

We see several key applications of this concept of similarity brushing to ease the interactive visual analysis of *n*-dimensional data as described below:

***n*D substructure brushing:** The most straight-forward application of similarity brushing, as already addressed in the example above, is the interactive selection of *n*D data substructures, which are nicely delimited in *n*-space (but not in *m*-space, i.e., in the visualization space). As described above, the procedure is to (1) select a visually well-separated core subset of the structure under investigation and then (2) extend this brush to also include the other (visually not so well-delimited) data items of the respective data substructure. The result of such an action can be seen in Figure 3.

$(n-k)$D subspace brushing: The substructure brushing described before does not necessarily have to consider the full dimensionality of the data set. Often there are features that reside inside a certain $(n-k)$D subspace of the original data domain but are lost if the whole set of dimensions is considered. The similarity brush allows for user-driven selection of dimensions to use to evaluate the similarity.

Interactive *n*D exploration: Another very useful application of similarity brushing is the interactive visual analysis of high-dimensional properties of the *n*D data. This can be achieved by interactively moving the visualization-based *m*D brush over the visualization and at the same time watching what data items get selected through the brush extension based on the *n*D distance metric. Examples of the application used to discover hidden relations are presented in Figure 4.

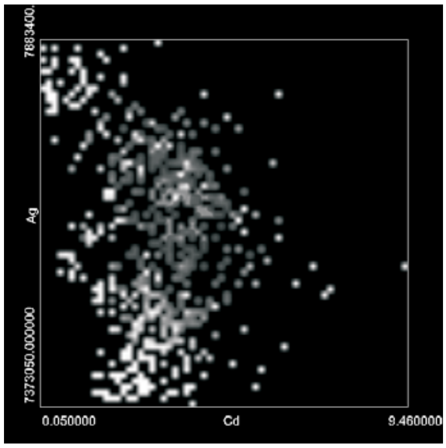


Figure 3: An *n*D similarity-based selection in the geochemical data [8] renders as sparse and scattered in 2D. It is even overlapped by different unselected items. Obviously it would be very hard and too laborious to select this structure using only conventional brushing.

Iterative brush refinement: The fourth interesting application of similarity brushing is the option for iterative brush refinements. In this application, the two-step process of similarity brushing are extended to form a process of alternately working in visualization and data space. For example, a brush can be started in visualization space as described above, then the brush can be extended to *n*D (again as above). But instead of stopping here, the user could go back to visualization space, e.g., alter the visualization setup by choosing a different visualization mapping $\mathbf{p}$ first and then again restrict the brush to only contain a subset of the currently selected data items (an AND operation with a second brush, for example).

Below, we now present a formal framework of how to integrate the selections in visualization space and those in data space.

3.2 Similarity Brushing – A Unified Framework

First we recall that we assume an *n*D data space D and an *m*D visualization space V, as well as a visualization transform $\mathbf{p} : D \rightarrow V$. In the following, we will now consider the two parts of similarity brushing, i.e., the visualization-based brushing as well as the data space based consideration of distances between data items.

For *m*D brushing (part 1), we assume that brushing interactions result in the assignment of a so-called degree-of-interest (DOI) function b_V to all the data items – $b_V(\mathbf{d}_i)$ is 1 if data item $\mathbf{d}_i$ is brushed, i.e., selected, and 0 if not. Often, b_V will be such a function to either map to 1 or 0, but nothing else (either a data item is brushed, or not). However, in many applications, it also makes sense to allow b_V to map to the en-

 ISBN 1213-6972 ISBN 80-86943-09-7

tire interval $[0,1]$ – called smooth brushing in the work of Doleisch et al. [6]. Even though we will in the meantime assume the b_V is either 0 or 1, we will further below demonstrate that all the here presented framework also works fine with a smooth brush b_V.

For nD extensions to our mD brushes, we assume to have a nD×nD metric $<.,.>_D \in \mathrm{R}^+$ available in data space to compute distances between nD data items (with $<\mathbf{d}_i,\mathbf{d}_j>_D = 0 \Leftrightarrow \mathbf{d}_i = \mathbf{d}_j$). In a first approach, we will consider the nD extension of an mD brush b_V to be defined as follows: all data items $\mathbf{d}_i$, which not yet are brushed by b_V, i.e., with $b_V(\mathbf{d}_i) = 0$, are checked whether there exists any other (brushed) data item $\mathbf{d}_j$, i.e., with $b_V(\mathbf{d}_j) = 1$, which is near enough, i.e., with $<\mathbf{d}_i,\mathbf{d}_j>_D < d_{\max}$. If such a near and brushed data item $\mathbf{d}_j$ can be found, then $\mathbf{d}_i$ is added to the brush.

In our unified framework, we formulate mD brushing and nD extensions of mD brushes as follows. In addition to brush b_V we assume a non-visual "brush" b_D to map nD distances to 1 (or 0), depending on whether the distance yields an inclusion within the extended brush (or not, respectively). We now integrate b_V and b_D to yield a combined brush b for all data items $\mathbf{d}_i$, depending on whether they are part of the extended similarity brush:

$$b(\mathbf{d}_i) = 1 - \min_j((1 - b_D(<\mathbf{d}_i,\mathbf{d}_j>)) + (1 - b_V(\mathbf{d}_j))) \quad (1)$$

In other words, to evaluate whether a data item $\mathbf{d}_i$ is part of the extended similarity brush b, all data items $\mathbf{d}_j$ are checked (at least in principle; in practice it is sufficient to check only those with $b_V(\mathbf{d}_j) > 0$ – all the others cannot generate a $b > 0$): If there is at least one data item $\mathbf{d}_j$ which (1) lies in the original brush, i.e., $b_V(\mathbf{d}_j) = 1$, and which (2) is near enough to data item $\mathbf{d}_i$, i.e., $b_D(\mathbf{d}_i,\mathbf{d}_j) = 1$, then also b is 1. This, of course, also holds if $\mathbf{d}_i$ itself lies in the original brush b_V. There are a number of nice properties of this integration to be mentioned:

Boundedness of b – The $((1-b_D(.)) + (1-b_V(.)))$-argument of the min is bounded (for an arbitrary j) between 0 and 2 (which potentially could lead to negative bs). But for $j = i$, $b_D(<\mathbf{d}_i,\mathbf{d}_j>) = b_D(0) = 1$. This yields that $((1-b_D) + (1-b_V))$ is bounded between 0 & 1 for $j = i$. Accordingly, the entire min-expression cannot become more than 1 which consequently yields that $0 < b < 1$.

Preservation of b_V – With the same line of argumentation as above we can show that $b(\mathbf{d}_i) \geq b_V(\mathbf{d}_i)$, i.e., for data items which already lie within the original brush, the extended brush cannot exclude them anymore.

Smooth brushing compliance – Equation (1) also holds for smooth brushes, i.e., $b_V \in [0,1]$ and $b_D \in [0,1]$. The $((1-b_D) + (1-b_V))$-expression can also be interpreted as a sum of two distances, one measured in visualization space $(1-b_V)$ and one measured in data space $(1-b_D)$. If the sum is small enough, then the resulting b can become greater than 0 which means that the respective point is included within the extended similarity brush b.

For the implementation, a number of optimizations can be realized, of course, to speed up the calculation of b. First, only those data items $\mathbf{d}_j$ need to be checked with a non-zero b_V (this most oftenly is a comparably small number). Second, data items $\mathbf{d}_i$, which are too far away from brush b_V after projection $\mathbf{p}$, do not need to be evaluated since they can never generate a $b > 0$. Practice shows that only a relatively small number of data items actually have to be checked to compute b.

4 SIMILARITY BRUSH WORKFLOW

The process of data exploration using the similarity brush is user-driven and constructed in a way that the user can take advantage of the automatic methods during the whole process. In the first stage a primary selection (depicted in yellow) is performed in the screen space using usual brushes and their various combinations (AND, OR, NOT). The secondary selection is a data-driven brush derived from the primary selection and is depicted in red. The last parameter is the similarity threshold that, basically, determines the extent of the selection.

The selection process can be iterated or refined to support complex data exploration tasks. Let's consider real world data (Figure 2). These data contain many outliers that for many reasons are usually considered an undesired feature of the data. To remove the outliers, we select several among them on screen and then extend this selection in the data space to include all the outliers. This selection can than be refined in other dimensions to include more outliers or remove entries that are of interest with respect to a different projection. After that the outliers, which are now selected using the similarity brush, can be removed and the data analysis can continue.

4.1 Advanced Interaction

To support the visual exploration tasks such as segmentation or classification a number of other functionalities is present. Any performed selection can be stored as a segment which excludes its entries from further brushing and is marked by a different color. In addition, every segment can be broken apart which reverses the segmentation process and returns its entries back to the data domain. With the support for unsharp and overlapping features provided by the similarity brush this allows for efficient user-based data-driven classification or segmentation of multidimensional data. (see example)

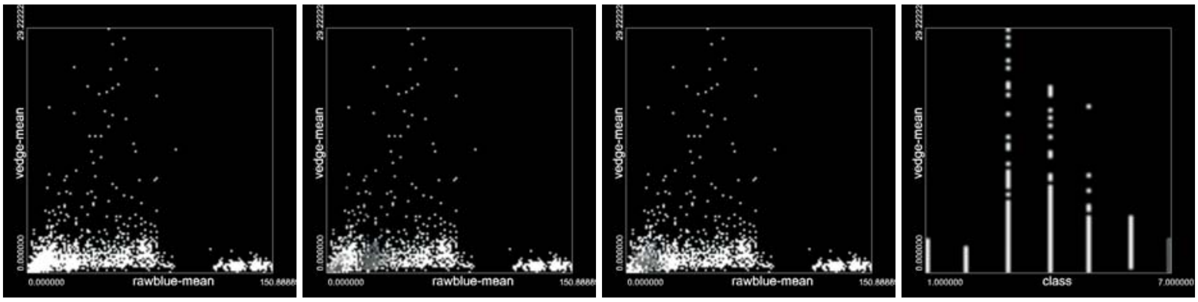


Figure 4: Major changes in the underlying structure discovered using the realtime exploration feature. The assumed indifferent region (first picture) in the bottom left of the scatterplot is evidently compiled of two separate structures (second and third picture.) The fourth picture proves the smaller structure being identical with the class number seven.

5 EXAMPLES

In the following sections, several examples illustrate the using of similarity brush to discover interesting multi-dimensional behavior or to easily perform complicated brushes. The data sets for these examples are the remote sensing data [19] obtained from SPOT satellites [17]. It contains 5 distinct channels (SPOT, magnetics, 3 bands of radiometrics) combined for a particular region in Western Australia. The second data set contains geochemical data [8] of concentration of multiple elements in a series of observed samples. The last one is one of the Statlog datasets [18] and contains samples produced by image processing together. It is a data set that is usually used for training automated techniques and thus it also includes classification information (brickface, sky, foliage etc.) We used this classification to partially evaluate relevance of our exploration.

5.1 Separate structures

Interesting topology can be discovered in a 2D display using the similarity brush. For example sudden changes in the brushes generated by two areas imply that these areas are separate structures in the original n-dimensional space. This can be explored using the realtime exploration feature of the similarity brush. The user moves the primary brush (often only a point selection) over the display and observes the changes of the secondary brush. An area in the image processing data, that was previously considered homogeneous, turned out to consist of two separate structures (Figure 4.) To illustrate the relevance of the data-driven brush we compared this new information to the classification provided with the data set. The entries encompassed in the brush fully correspond to those segments of the source images that depict grass.

5.2 Subspace relations

Unlike the previous example, the structures don't only have to be separate in the full dimensionality of the original data domain. Two different three-way combinations (SPOT, Magnetics, Uranium and SPOT, Thorium, Uranium) of dimensions were used to create two different similarity measures. Given the same screen-based primary selection (samples with very high potassium values) two data-driven selections were derived from that (Figure 5) and we observe differences between them. The most significant difference is that a change in the set of considered dimensions splits the dense U-shaped cluster into two, revealing its two-fold intrinsic nature. The left part (with low thorium values, evaluated using SPOT, Magnetics, Uranium) and the right part (with high thorium values, evaluated using SPOT, Thorium, Uranium). The left part is much more similar to the samples with high potassium values with respect to the magnetics characteristics. Unlike that, the right part is more similar to the high potassium samples with respect to the concentration of thorium.

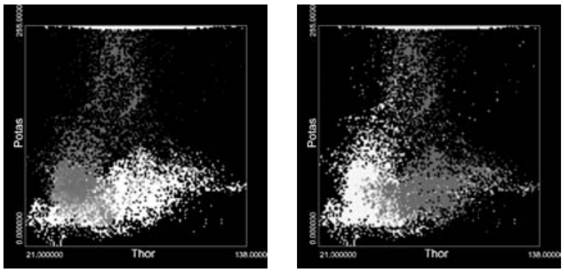


Figure 5: Two different subsets of dimensions were used to compute similarity information. This resulted in two different data-driven brushes (red) given the same primary selection (yellow).

Using only usual visualization the cluster would probably be considered homogeneous. The real nature of the cluster could be discovered using automatic data mining, but without user interaction the analysis of such a knowledge would require additional human-based effort.

5.3 Anomalies

The geochemical data contains an interesting entry that was discovered using the similarity brush. When investigating the data set using the realtime similarity brush, one entry was found to generate no secondary brush. Even though this entry is not depicted as an outlier (in

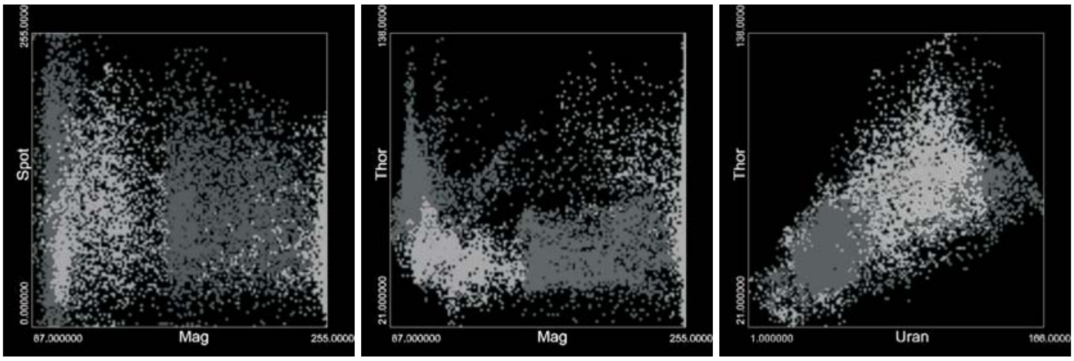


Figure 6: Complex, unsharp and overlapping segments are seldom feasible when conventional brushing techniques are applied. With similarity brush the segmentation of the remote sensing data set took less than a minute and required only simple interaction. Please refer to [23] for full color pictures.

any projection of the geochemical data) it is not similar to any of the remaining entries unless a very loose threshold is chosen. It is a hot candidate for a multidimensional outlier – a sample that lies within a reasonable range of neighbors in every projection, but the sets of the neighbors change over the dimensions (imagine a point in the centre of a hollow sphere-shaped shell.) This makes it isolated if the full dimensionality of the space is taken into account.

5.4 Interactive segmentation

Automated techniques are often used to perform segmentation tasks. But the automated techniques in many ways benefit from the domain knowledge, intuition and abstract thinking of the human user. In conjunction with the similarity brush, the user has the ability to incorporate his capabilities into the segmentation process by specifying interactively the core of the segments and the difference tolerance level within a segment. This gives him a two-fold advantage over the automated techniques. First, the user can specify complex and sophisticated starting points for the segmentation process via creating the primary selection. Second, the user can at any time refine his selections, backtrack the steps and change the decisions he/she made. These actions are rarely performed in an automatic segmentation process.

In addition, the similarity brush interleaves the semantic identification process with the segmentation process. If a computer performs data segmentation automatically, it often produces segments without an actual meaning and additional human-based processing has to take place in order to identify the semantics of the segments. In interactive segmentation provided by similarity brushing the segment starts as a specific core that is user-specified and thus correspond to some real world knowledge provided by the human.

The Figure 6 shows the results of user-based segmentation on a dense multidimensional data set. The resulting segments are consistent and prove to be compact in all views (only three are depicted here though.) The segments have sparse boundaries and overlap in most of the views, which is a common property among real world multivariate data, and would be difficult to mark out using only conventional screen-based brushes.

6 EXTENDING THE CONCEPT

The similarity information computed from all the dimensions of a data set offers hints about the *n*D nature of the information. As a concept, this can be easily incorporated into other popular displaying techniques, such as the parallel coordinates or the histograms. Also possible extensions of the concept could be used in scientific or flow visualization.

The design allows for arbitrary similarity functions to be used. Among the most popular ones are the spatial distance measures (Euclid, Chebychev, Manhattan, Mahalanobis.) Another option is to use information gained from e.g. fuzzy clustering or other automated data mining techniques. Such techniques detect items of similar properties in the set and group them together. The similarity of two samples could thus be evaluated using this information.

Another promising extension is to allow new samples to "join" the screen-based brush if they are close enough or follow other given criteria. This would allow for the chaining effect known from data mining and structures of even more complex shapes could be addressed.

7 CONCLUSION

The tool presented in this paper uses a combination of visual and automatic data-mining to introduces a new way to integrate *n*-dimensional information into a low dimensional display. By interaction with the similarity brush, the user gets to directly touch the multidimensional structures in their original space instead of having to only approximate this by numerous low-dimensional actions. This interaction technique can be used to enhance visual exploration of multidimensional data. As shown by the examples, complex multidimensional topology can be observed even in a simple scatterplot by using the similarity brush. With the use of the similarity brush for visual exploration, extra information can be provided that might help the user to steer his precious attention in further visual exploration actions.

This intuitive tool does not encumber the user's perception by generating visual overload and can be successfully used in many displays. We believe that the similarity brush may well become a useful interaction tool for exploring multidimensional data in many future applications.

8 ACKNOWLEDGMENTS

The research is a part of the basic research on visualization conducted in cooperation between the Comenius University, Bratislava and the VRVis Research Center in Vienna (http://www.vrvis.at). VRVis is funded by an Austrian research program called Kplus. The project is also partly funded by the Slovak government grant VEGA 1/3083/06.

The authors would also like to thank Prof. Peter Filzmoser from Vienna University of Technology for his kind help with the data sets.

REFERENCES

[1] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: an implementation and evaluation. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 619–626, New York, NY, USA, 1992. ACM Press.

[2] M.Q.W. Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings of the working conference on Advanced visual interfaces*, pages 110–119. ACM Press, 2000.

[3] R.A. Becker and W.S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[4] E.A. Bier, M.C. Stone, K. Pier, W. Buxton, and T.D. DeRose. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, 1993. ACM Press.

[5] H. Chen. Compound brushing. In *Proceedings of the IEEE Symposium on Information Visualization 2003 (INFOVIS'03)*, 2003.

[6] H. Doleisch and H. Hauser. Smooth brushing for focus+context visualization of simulation data in 3D. *Journal of WSCG*, 10(1):147–154, 2002.

[7] S. Eick and G. Wills. *High Interaction Graphics*. 1995.

[8] C.Reimann et al. *Environmental Geochemical Atlas of the Central Barents Region*. 1998.

[9] Y.H. Fua, M.O. Ward, and E.A. Rundensteiner. Navigating hierarchies with structure-based brushes. In *INFOVIS*, pages 58–64, 1999.

[10] I. T. Jolliffe. *Principal Component Analysis*. Series in Statistics. Springer-Verlag, 1986.

[11] T. Kohonen. *Self organizing maps*. Springer, New York, 2000.

[12] R. Kosara, H. Hauser, and D. Gresh. An interaction view on information visualization, star. In *EUROGRAPHICS 2003*, 2003.

[13] J. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, 1978.

[14] A.R. Martin and M.O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 271, Washington, DC, USA, 1995. IEEE Computer Society.

[15] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, 2004.

[16] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, New York, NY, USA, 1995. ACM Press.

[17] http://www.spot.com.

[18] http://www.liacc.up.pt/ml/statlog/datasets.html.

[19] http://davis.wpi.edu/˜xmdv/datasets.html.

[20] E.R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[21] M. O. Ward. Creating and manipulating n-dimensional brushes. In *Proceedings of Joint Statistical Meeting*, pages 6–14, August.

[22] M.O. Ward. Xmdvtool: integrating multiple methods for visualizing multivariate data. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 326–333, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[23] http://www.vrvis.at/via/research/simi-brush/.

[24] G. J. Wills. Selection: 524,288 ways to say "this is interesting". In *INFOVIS '96: Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, page 54, Washington, DC, USA, 1996. IEEE Computer Society.

3D Reconstruction and Visualization of Spiral Galaxies

Kristian Hildebrand
MPI Informatik
Stuhlsatzenhausweg 85
Saarbrücken, Germany
khildeb@mpi-inf.mpg.de

Marcus Magnor
Technical University of Braunschweig
Muehlenpfordtstrasse 23
Braunschweig, Germany
magnor@mpi-inf.mpg.de

Bernd Fröhlich
Bauhaus University Weimar
Bauhausstrasse 11
Weimar, Germany
bernd.froehlich@medien.uni-weimar.de

ABSTRACT

Spiral Galaxies are among the most stunning objects in the night sky. However, reconstructing a 3D volumetric model of these astronomical objects from conventional 2D images is a hard problem, since we are restricted to our terrestrial point of view. This work consists of two contributions. First, we employ a physically motivated, GPU-based volume rendering algorithm which models the complex interplay of scattering and extinction of light in interstellar space. Making use of general galactic shape information and far-infrared data, we secondly present a new approach to recover 3D volumes of spiral galaxies from conventional 2D images. We achieve this by an analysis-by-synthesis optimization using our rendering algorithm to minimize the difference between the rendition of the reconstructed volume and the input galaxy image. The presented approach yields a plausible volumetric structure of spiral galaxies which is suitable for creating 3D visualization, e.g., for planetarium shows or other educational purposes.

Keywords
3D visualization, astronomical visualization, 3D reconstruction

1 Introduction

The night sky emanates a deep fascination. It has been the target of contemplation and research efforts since the earliest beginnings of human culture. The colorful, attractive appearance of astronomical objects is awe-inspiring. Besides their esthetical value, astrophysicists are able to draw conclusions about the origin of the cosmos from scientifically studying these objects. Therefore, telescopes around the world and in space record and collect data that is not only useful for physicists but also stunning and beautiful to everybody on earth.

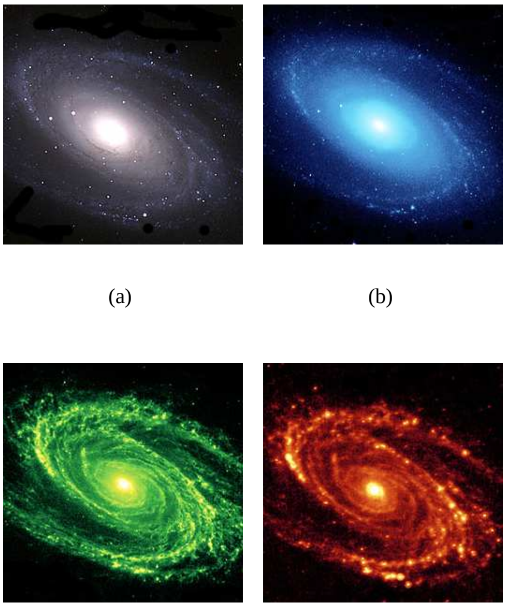
By looking through an eye-piece of a telescope one can see various objects in the night sky, e.g. planets, stars, all sorts of nebulae and galaxies. To get an impression on how these objects look like from a different viewpoint than our terrestrial position one has to visualize astronomical objects in their three-dimensional shape and simulate the visual effects by exploring physical properties of light in interstellar material.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006, January 30-February 3, Plzen, Czech Republic.

The increasing interest of 3D visualization in TV documentaries, science fiction movies, games and education makes it desirable to find a realistic representation of astronomical objects. Todays animations are often based on a more artistic then physically correct representation of astronomical objects, even though it is useful to give a more realistic impression of what can be observed. Furthermore we can use our results to provide additional information in telescope applications like recently published by Linţu et al. [LM05] to increase the understanding of the observed data.

In this paper we present an approach on how to determine the three-dimensional shape of *spiral galaxies* from conventional 2D images, which is, in general, a very challenging problem, due to our restricted point of view.

However, our approach relies on several physical information about the object, like a general shape evolved from its formation. Additionally we use different band-filtered data from the objects observation to gain more insights of the material it consists of. The appearance of *spiral galaxies* is significantly determined



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

Figure 1: M81 as photographed from the Spitzer Space Telescope in the infrared light ©NASA/Spitzer [Tel]. (a) Galaxy M81 in visible light. (b), (c), (d) M81 at three different infrared wavelengths (3.6, 8, and 24 μm, respectively).

by the amount of interstellar material within the galaxies, because clouds of dust have scattering and extinction properties at different visible-light wavelengths as can be seen in Figure 4 from [BM98]. To be able to reconstruct a 3D volumetric model of *spiral galaxies* we have to simulate these effects.

The paper is structured as follows. After a brief overview of related work, we provide some fundamental background information about galaxies in Section 3 We describe our volumetric radiance model, which is important for the reconstruction process in Section 4 and illustrate in detail how to recover the actual 3D volumetric shape in Section 5 Our results are presented in Section 7

2 Previous Work

Pioneering work in visualizing virtual space journeys was done by Jim Blinn [Bli87] and his set of animations called 'Voyager Fly-by Animations', 'Cosmos' and 'The Mechanical Universe' from the late 70ies and early 80ies. Many spectacular, almost always artistic, 3D fly-throughs of astronomical objects can be seen in planetariums. An exceptionally physically based work is done by Nadeau et al. [NGN$^+$01], [NE]. They employed massive computational power to create scientifically justified views of the orion nebula. Their visualization relies on a 3D model of the Orion nebula that was determined by astronomers from various observational data [ZO95]. Hanson et al. [HFW00] did a lot of work on large scale visualization of astronomical data and more recently on exploring the physical Universe as an enormous environment. They introduce a so-called powers-of-ten visualization architecture to provide scale-independent modeling and rendering [FH05].

Magnor et al. [MKH04] recently reconstructed and rendered 3D models of planetary nebulae. Former astrophysical research had shown that some planetary nebulae have specific symmetry characteristics due to physical processes of their formation. The basic idea is to use astronomical image data and symmetric structural constraints of the nebulae to reconstruct the three-dimensional volume by an analysis-by-synthesis approach. They introduced the term *constraint inverse volumetric rendering* (CIVR) as a GPU-based optimization procedure to reconstruct a volumetric model for planetary nebulae. Magnor et al. [MHLH05] most recently visualized physically correct *reflection nebulae* by taking into account the astrophysical properties of dust in interstellar space. *Reflection nebulae* are clouds of interstellar dust which are reflecting visible light of a nearby star or stars. That results in a very colorful interplay between scattering and extinction effects and makes them one of the most colorful objects in the night sky. The used volume rendering approach employs the Henyey-Greenstein scattering phase function to create a lookup table for the amount of light scattered in the observer's direction [HG41], [Gor04], [HG38]. The synthetic data sets mimic artificially generated reflection nebulae very realistically.

In this paper we rely on the proposed reconstruction and rendering technique ([MKH04], [MHLH05]) but employ different optimization constraints and an adopted visualization model.

Fundamental basic knowledge and research about galactic astronomy can be found in Binney et al. [BM98]. They provide a complete overview of colors, morphology and photometry of galaxies, as well as the properties of interstellar material and its effect on observed data.

3 Background

A galaxy is a system of stars, interstellar gas and dust, dark matter in the center and possibly dark energy. Galaxies usually contain 10 million to one trillion stars orbiting a center of gravity. It consists of rarefied interstellar material, star clusters, single stars and various types of nebulae, such as emission-, dark-, planetary- and reflection nebulae. The generic shape in Figure 3 can be divided into a center bulge embedding very old stars, a circular disk of younger stars and a surrounding spherical halo [BM98].

3.1 Shape Classification

Astronomers classify galaxies based on their overall shape and further by the specific properties of the individual galaxy, like the number of spiral arms, the degree of the ellipse or the pitch angle of the spiral. The system of galaxy classification is known as the Hubble sequence or Hubble tuning fork which is shown in Figure 2. This classification scheme starts at the left with *elliptical galaxies* (E0-E6 types) divided by the factor of oval-shape. Then the diagram splits into two branches. The upper branch shows *spiral galaxies* (Sa-Sc types) which are basically split into different spiral pitch angles. The lower branch (SBa-SBc types) covers *barred-spiral galaxies* that differ in their characteristic formed bar in contrast to the spherical shaped bulge of Sa-Sc types. We will focus on recovering galaxies of type Sb and Sc using data from the Spitzer telescope [Tel], in particular Galaxy M81, a publicly available data set that can be used for the proposed reconstruction method.

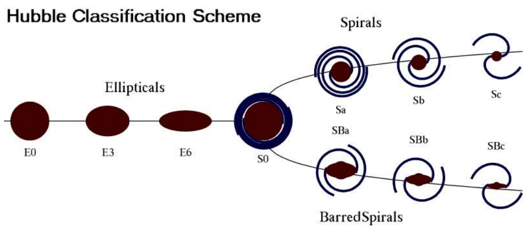


Figure 2: Hubble Classification Scheme ©S.D. Cohen (from [Coh]). The Hubble Classification Scheme is able to capture the topological diversity of most galaxies. From left to right, spherical and elliptical shapes are succeeded by spirals of different extent, with or without a central bar.

The arms of spiral galaxies approximately have the shape of a logarithmic spiral and are areas of high density or density waves and can be observed at visible wavelengths. The high concentration of gas and dust in the arms facilitates star formation of very bright stars.

3.2 Interstellar Dust

A galaxy viewed from the front, the so-called face-on view, shows dark spiral stripes containing interstellar dust (see Figure 3). Interstellar dust has the property to scatter and absorb photons at different wavelength. Blue light is scattered more often, because the size of many of the individual grains of space dust is about the same as the wavelength for blue light varying between $100nm$ and $1\mu m$. It means, that much of the blue light emitted from stars in the galaxy behind the dust clouds gets scattered away from our direct view, making the stars in the galaxy, as we see them through the dust look redder and dimmer than they actually are [BM98]. The wavelength-dependent scattering properties of particles are described by the Mie scattering theory [vdH82]. Furthermore photons that get absorbed from the dust convert the energy into heat. Thus, the dust transforms blue light into far infrared light and the absorption of starlight warms dust grains to $\approx 10K$. At this temperature they radiate significantly at $\lambda \approx 200\mu m$, and photons of this wavelength can escape.

Figure 1 shows infrared images obtained by Spitzer's infrared array camera, a space telescope to obtain images and spectra in infrared at wavelengths between 3 to 180 micron, that cannot be detected from Earth [Tel]. It exhibits a four-color composite of Galaxy M81 which is located at a distance of 12 million light-years from Earth. The images of visible and invisible light show emissions from wavelengths of 3.6 microns (blue), 8 microns (green), and 24.0 microns (red). Images in near-infrared collected at 3.6 micron trace the distribution of older and redder stars and are virtually unaffected by obscuring dust. As one moves to longer wavelengths, the spiral arms become the dominant feature of the galaxy. The 8 micron emission is dominanted by infrared light radiated by hot dust that has been heated by nearby luminous stars. The dust particles are composed of silicates, carbonaceous grains and polycyclic aromatic hydrocarbons and trace the gas distribution in the galaxy. The well mixed gas and dust, which is best detected at radio wavelengths, provide a reservoir of raw materials for future star formation. The 24-micron image shows emission of warm dust heated by the most luminous young stars. The bright knots show where massive stars are being born. These star formation regions are of great astrophysical interest because they help identifying the conditions and processes of star formation [Tel].

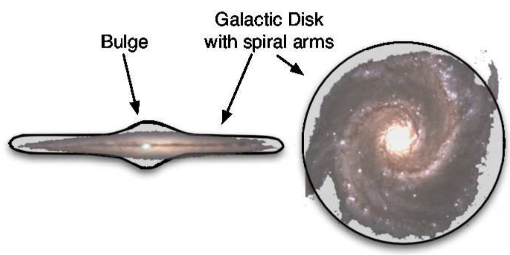


Figure 3: Generic Galaxy Shape. On the left is the so-called edge-on view (90^o). The dust is concentrated along a galaxy's equatorial region. Compared to the right, which is often referred to as face-on view (0^o), where the dust is concentrated along the spiral arms.

 ISBN 1213-6972 ISBN 80-86943-09-7

4 Galaxy Visualization

From our terrestrially confined viewpoint, recovering the actual three-dimensional shape of distant astronomical objects is very challenging. The key to reconstruct a 3D *spiral galaxy volume* from an 2D image is to find a visualization technique which is able to reproduce the visual effects of the galaxies appearance. Figure 4 shows that due to the distribution of dust within the galaxy a complex interplay of light and dust takes place. The appearance of galaxies as we see them can be described by effects of scattering and extinction [BM98]. Thus, we need to find a visualization model that simulates realistically these effects.

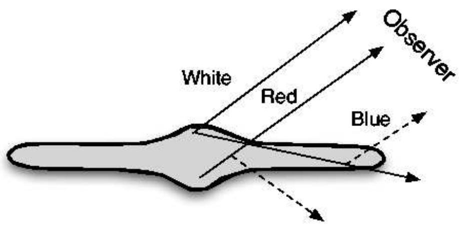


Figure 4: Effects of scattering and absorption of light by dust. Light from the top reaches the observer without obstruction. Light from the lower part is partially absorbed and scattered into the path of the observer.

4.1 Volumetric Radiance Model

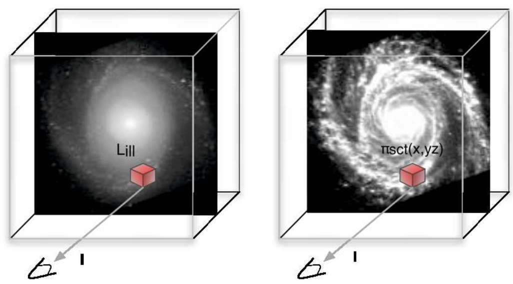
At this point we can make use of a method proposed by Magnor et al. [MHLH05]. They are simulating the effects of interstellar dust for *reflection nebulae*. Adopting their visualization model to galaxies can be done as follows: first, we subdivide the space around the galaxy into voxels. Each voxel is assigned the radiance L_{ill} arrived at the voxel from the stars and a value proportional to the density of dust π_{sct}, where $\pi_{sct} = \sigma_{sct} \cdot l$, with l as the size of the voxel and σ_{sct} as the scattering coefficient, the average amount of scattering, for a voxel.
Only a fraction $P(\pi_{sct})$ of the radiance L_{ill} at a voxel is scattered into the observers direction to define L_{sct}. $P(\pi_{sct})$ is pre-computed and tabulated using a Monte-Carlo simulation. Since we assume isotropic scattering due to the large-scale structure, the portion of scattered light is not direction-dependent in contrast to reflection nebulae rendering [MHLH05].

$$L_{sct} = L_{ill} \cdot P(\pi_{sct}) \quad (1)$$

However, the scattered light still must travel the path form the voxel to the observer, where it also is attenuated along the line of sight due to optical depth $\pi_{opt} = \pi_{sct}/a$ in the interstellar medium. The albedo, $a = [0, 1]$ can be described as the average percentage of radiation that is being scattered on a single dust particle. It becomes zero when the dust is completely black and all incident radiation is absorbed and one when all photons are deflected by the particle. As pointed out in [Gor04], it is reasonable to assume that for the current measured and analyzed data at visible wavelengths the albedo is $a \approx 0.6$ throughout the galaxies. That yields

$$L = L_{sct} \cdot exp^{-\int_0^l \pi_{opt}(l')dl'}. \quad (2)$$

Since π_{sct} and π_{opt} vary with wavelength we compute L separately for the red, green and blue channel. Binney et al. [BM98], Magnor et al. [MHLH05] and Cardelli et al. [CCM89] illustrate the wavelength-dependent effects of extinction which can be described by astrophysical parameters such as the *ratio of total-to-selective extinction*. We also take these values into account.



(a) (b)

Figure 5: Discretization of the space around the galaxy into voxels. Each voxel emits light according to local star density (a). A fraction of the light is scattered depending on dust density or scattering depth π_{sct}. More light is absorbed on distance l from the voxel to the observer.

Figure 5(a) shows the radiance L_{ill} that is received at a volume position (x,y,z) and the scattering depth π_{sct} proportional to the dust density in Figure 5(b) . More details on how we can derive these values from the 2D observations for reconstruction are described in Section 5

4.2 Interactive GPU-Raycasting

Our rendering algorithm relies on graphics hardware-based ray-casting. The basic idea, proposed by Krueger [KW03], is simple. The dataset is stored in a 3D texture to take advantage of built-in trilinear filtering. Then a bounding box geometry is created encoding the position in the data set as colors, i.e., we can interpret these as 3D texture coordinates.
The rendering algorithm runs in four passes. The first two passes prepare the proxy-geometry, i.e., render

 ISBN 1213-6972 ISBN 80-86943-09-7

front- and back-faces, and compute the ray direction and length. In the third pass we first issue a fragment program to step along the viewing vector from front to back in voxel length intervals. The pre-computed scattering table $P(\pi_{sct})$ is uploaded as a 1D floating-point texture to graphics memory. A 3D floating point texture stores scattering depth π_{sct} and illumination L_{ill} for each voxel. At each step along the ray we lookup local scattering depth π_{sct} and voxel illumination L_{ill} from the 3D volume textures. These values are trilinear interpolated on graphics hardware. Then the fragment program queries the scattering lookup table to determine $P(\pi_{sct})$. L_{sct} can be determined by (1). L_{sct} undergoes extinction $\sigma_{ext} = \pi_{sct}/a$ on the line of sight which is accumulated by stepping along the ray to compute L (2). The image correction is done in the final fourth pass.

5 3D Reconstruction

The proposed visualization model is able to capture the general effects of interstellar material that are responsible for the overall appearance of galaxies. However, to reconstruct the galaxy from a conventional 2D image we have to recover dust density and light distribution information from observational data.

Before analyzing the images in Figure 1 we compute their geometric moments up to the second order and center, rotate and de-project the images, i.e., correct to a face-on view as commonly done in astrophysical research. Now, we can recover approximately the density of dust by adding up Figure 1(c) and Figure 1(d). Adding up the dust images is essentially like accumulating different sorts of interstellar material that radiate at different infrared wavelengths because of their size and temperature. The image intensity can be then interpreted as a dust density which is proportional to the scattering depth π_{sct}, shown in Figure 6(b). The radiance L_{ill} received at a voxel can be taken from Figure 6(c) which shows the star light unaffected by obscuring dust and can be interpreted as the distribution of light at a specific point in the galaxy.

However, we still have to take the amount of light into account that gets scattered in the observer's direction. Also, we account for the attenuation by optical depth when light travels through the galaxy to the observer as seen in Eq. (1) and (2). L_{ill} and π_{sct}, taken from the observational data, are sufficient to define our volume data structure.

We now can try to fit our 2D image in the generic galaxy shape using a simple back-projection approach. By back-projecting the image we smear it through the volume constraint by the generic galaxy shape which can be described by a gaussian function (see Figure 3). We discard all values outside the model and weight their contribution according to the distance from the center.

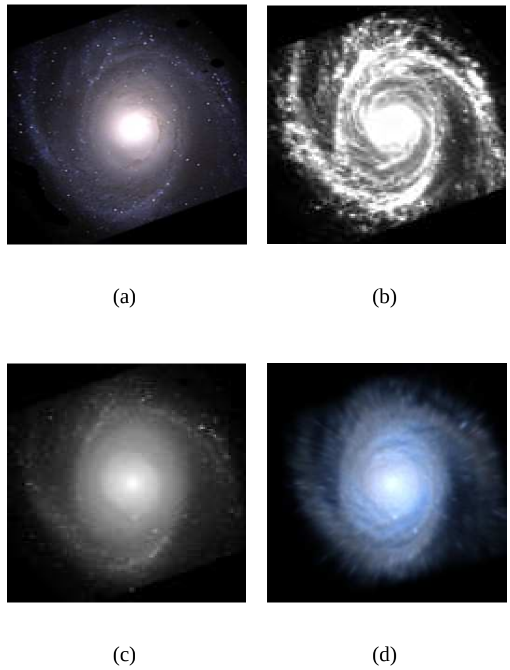


Figure 6: Reconstruction results from infrared image data, see Figure 1. We can achieve similar rendering results by extracting a dust distribution from the infrared images. (a) Original image in visible light, deprojected to face-on view. (b) Reconstructed dust distribution (c) Star light distribution at $3.6\mu m$ (d) Resulting rendition.

However, just back-projecting the dust distribution into the generic shape creates stripes that make the image look unnatural. Using procedural noise [Per85] we can break up these stripes in a more natural way.

Figure 6 shows, that the original image (a) can be implicitly reconstructed by using the dust distribution (b) and a radiance profile in image (c). Image (d) shows the result. It is interesting to see that without any further information, just by using dust and light distribution we can achieve a similar appearance of the galaxy. The blueish tint of the rendition (d) shows the assumed dust density map is not sufficient enough to reconstruct the galaxy realistically.

Using the adopted *reflection nebulae* rendering [MHLH05], [Hil05] is still an approximation, since the effects of scattering and absorption effects for the nebulae are only evaluated locally for a few stars and not globally for the entire galaxy. However, we can assume a dust density or scattering depth π_{sct} and a radiant power of star light L_{ill} at any voxel for our galaxy visualization model. By reconstructing a dust density map from several band-filtered infrared images it is possible to find a valid representation for an original image. As mentioned in Section 3.2 mid- and far-infrared data provide necessary information about the dust distribution of the galaxy. Now, we can expand this approach by using the reconstructed dust density map as an initialization parameter for an analysis-by-synthesis algorithm to approximate the original shape more closely.

6 Analysis-by-Synthesis Reconstruction

3D image analysis-by-synthesis is the general concept of inverting the image formation process by solving the forward problem repeatedly while adjusting the parameters of the reconstruction until the differences between the original and the synthesized image are minimized.

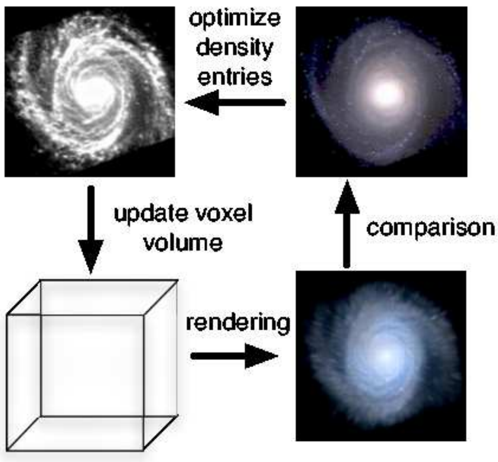


Figure 7: Analysis-by-synthesis scheme for spiral galaxy reconstruction. The volume is rendered and the difference between rendition and original is evaluated using sum-of-squared differences. After optimizing the model parameters in the dust density distribution we update the volume and render again.

The *spiral galaxy* reconstruction approach relies on constraint inverse volume rendering (CIVR), Magnor et al. [MKH04]. However, our CIVR approach is based on the generic shape of the galaxy and the proposed visualization method. The model we want to optimize is the dust density map which is proportional with the scattering depth π_{sct}. The approximated image in Figure 6(b) can be used as prior knowledge and as an initial guess for the optimization. The galaxy rendering provides the basis of our approach, since we assume that it is a plausible way to realistically visualize galaxy volumes. It is important to understand that a visualization which cannot provide plausible renderings, cannot be used for this approach, since we rely on evaluating the error functional based on the rendition and the image difference.

Given that the galaxy visualization is a non-linear process we employ non-linear optimization, i.e., a standard implementation of Powell's non-linear optimization method [PFTV92]. *Powell's direction set* numerically evaluates the error function's local slope along all dimensions of the model $m_{1...N}$ from which it determines the *conjugate-gradient* direction.

The aim of the optimization is to determine the closest possible solution for the 2D projection, i.e., the volume rendering result that matches as closely as possible with the original galaxy image at visible wavelengths. Each optimization iteration step entails a modification in the volume data set, uploading the modified data onto the graphics card, rendering the model again and re-evaluating the error measure, as shown in Figure 7. To qualify the difference between both images we compare the corresponding pixel using the sum-of-squared-differences (SSD)

$$\arg \min_{d_{1...N}} \sum (p(x,y) - p_r(x,y))^2 \qquad (3)$$

where $d_{1...N}$ denotes the parameters in the dust density map and the color parameters for the overall approximated star colors. Additionally, the error functional penalizes negative values and scattering depth π_{sct} values that reach outside the scattering table for values $\pi_{sct} > 10.0$. The color values are also penalized, if they fall outside the RGB range. That allows us to constrain our optimization to physically reasonable values. Magnor et al. [MKH04] employed several error functionals of which the SSD error measure yielded the fastest convergence. The algorithm table summarizes the steps again:

Algorithm 1 Analysis-by-Synthesis

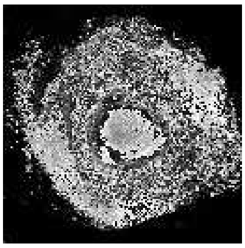
Back-project illumination and dust density map into generic shape;
Render galaxy volume;
Initialize optimization parameters, i.e., 2D scattering depth;
while Convergence not reached **do**
  Render volume;
  Optimization using Powell's Direction Set;
  Compute SSD to evaluate error;
  Penalize parameters that are out of range;
  Update optimization parameters, i.e., 2D scattering depth and color values;
  Back-project and update 3D volume on the graphics card;
end while

The algorithm stops, when the difference between optimization steps is lower than a certain tolerance value. From the optimization point of view this approach underlies a high-dimensional parameter space. Each iteration step we modify our parameters until the algo-

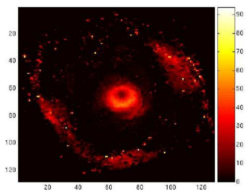
rithm converges to a minimum of the error function. Since we are dealing with a non-linear optimization problem, a global convergence to the global minimum can not be guaranteed.
If the initial guess is not close to the global minimum, or the parameters are not reasonable constraint, the algorithm does not converge to a physically plausible solution. Also, if our rendering procedure does not map the values closely to the original projection a convergence cannot be expected, because we are not able to produce the desired values.

(a)

(b)



(c)



(d)

Figure 8: Figure (a) shows the original M81 image deprojected to face-on view. (b) M81 after analysis-by-synthesis optimization. (c) Reconstructed dust density map.(scaled for displaying purposes) (d) Difference image between original and optimized image.

7 Results

Figure 8 shows that the reconstructed image (b) closely resembles the original galaxy photograph (a). The proposed visualization model is able to recover the overall appearance of galaxies. The dust density distribution entries are optimized which is a sufficient model to reconstruct the volume. Despite a reasonable guess for our analysis-by-synthesis procedure the optimization computation took about three days on a 3.0Ghz Pentium4 with nVidia GeForce 6800 Ultra graphics board.
Figure 8(d) shows the difference between original and reconstruction. One can see that especially the galactic bulge area exhibits high differences. That is because the dust consistency changes drastically throughout the galaxy center due to very hot stars. The reconstructed dust density distribution varies compared to initial guess in Figure 6(b) and is very noisy. The reason is that there are many regions where dust density changes, e.g. because of star formation.

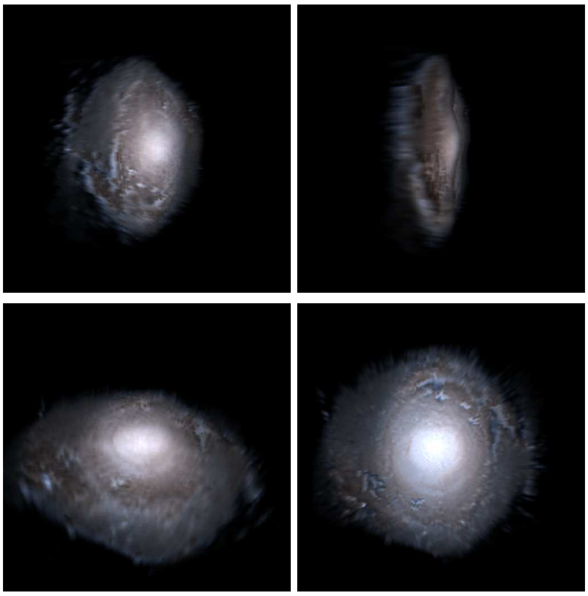


Figure 9: Resulting fly-by images for the presented 3D volume reconstruction of Spiral Galaxy M81.

Figure 9 shows that the proposed method recovers an approximated, plausible shape of the volume using the galaxy rendering, which yields the conclusion, that the proposed method is a promising approach to attack the problem.
However, there are still a lot of problems, mostly related to approximations in the visualization model. The radiance map, as seen in Figure 6(c), doesn't have enough influence on the rendering to display the bluish areas along the spiral arms. That should be taken into account during optimization. Furthermore, we should take other physical parameters into account to represent the galaxies appearance more closely.

8 Conclusion

We have presented an adopted rendering method to reconstruct a plausible shape for *spiral galaxy M81*. Its inherent generic shape and additional observations in far-infrared enable us to use a model to describe the galaxy's three-dimensional dust distribution in space, thereby constraining the reconstruction problem. By rendering realistic images of our model and comparing the rendering results to the original image data, we employ an optimization approach that helps converge towards a reasonable dust density distribution for the galaxy. Using the optimized model we are able to closely resemble a realistic appearance of the galaxy by fitting the values to a generic shape. Figure 9 shows a series of images from different view points around galaxy M81.

References

[Bli87] J. Blinn. Voyager fly-by animations. *Animations depicting space missions to Jupiter, Saturn and Uranus*, 1977-87.

[BM98] J. Binney and M.Merrifield. Galactic astronomy. *Princeton University Press*, 1998.

[CCM89] J. Cardelli, G. Clayton, and J. Mathis. The relationship between infrared, optical and ultraviolet extinction. *Astrophysical Journal*, pages 345:245–256, 1989.

[Coh] S. D. Cohen. Hubble classification scheme. *website:* `http://en.wikipedia.org/wiki/Image:Hubblescheme.png`. *visited August 2005*.

[FH05] Chi-Wing Fu and A. Hanson. The powers-of-ten visualization architecture for exploring the physical universe. *submitted for publication*, 2005.

[Gor04] K. Gordon. Interstellar dust scattering properties. in a. witt, g. clayton, and b. draine, editors. *Astrophysics of Dust. ASP conference series*, 2004.

[HFW00] A.J. Hanson, Chi-Wing Fu, and E.A. Wernert. Very large scale visualization methods for astrophysical data. *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization*, pages 115–124, 2000.

[HG38] J. Henyey and J. Greenstein. The theory of the colors of reflection nebulae. *Astrophysical Journal*, 88:580–604, 1938.

[HG41] J. Henyey and J. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 93:70–83, 1941.

[Hil05] K. Hildebrand. Rendering and reconstruction of astronomical objects. *Diploma Thesis*, 2005.

[KW03] J. Krueger and R. Westermann. Acceleration technique for gpu-based volume rendering. *Proceedings of IEEE Visualization*, pages 287–292, 2003.

[LM05] A. Lintu and M. Magnor. Augmented astronomical telescope. *Second GI-Workshop VR/AR Proceedings*, 2005.

[MHLH05] M. Magnor, K. Hildebrand, A. Lintu, and A. Hanson. Reflection nebula visualization. *Proc. IEEE Visualization*, pages 255–262, 2005.

[MKH04] M. Magnor, G. Kindlmann, and H. Hansen. Constrained inverse volume rendering for planetary nebulae. *Proceedings of IEEE Visualization*, pages 83–90, 2004.

[NE] D. Nadeau and E. Engquist. Volume visualization of the evolution of an emission nebula. *website:* `http://vis.sdsc.edu/research/hayden2.html`. *visited January 2005*.

[NGN+01] D. Nadeau, J. Genetti, S. Napear, B. Pailthrope, C. Emmart, E. Wesselak, and D. Davidson. Visualizing stars and emission nebulae. *Computer Graphics Forum*, 20:27–33, 2001.

[Per85] K. Perlin. An image synthesizer. *Proceedings of ACM SIGGRAPH*, pages 287–296, 1985.

[PFTV92] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. Numerical recipes in c. *Cambridge University Press*, 1992.

[Tel] Spitzer Space Telescope. *website:* `http://www.spitzer.caltech.edu/`. *visited June 2005*.

[vdH82] H. van de Hulst. Light scattering by small particles. *Dover Publications Inc., New York*, 1982.

[ZO95] W. Zhen and C. ODell. A three-dimensional model of the orion nebula. *Astrophysical Journal*, 438:784–793, 1995.

Real-time Plane-Sweep with local strategy

Vincent Nozick Sylvain Michelin Didier Arquès

SISAR team,
Marne-la-Vallée University, ISIS Laboratory,
6 cours du Danube, France, 77 700 Serris
{vnozick,michelin,arques}@univ-mlv.fr

ABSTRACT

Recent research in computer vision has made significant progress in the reconstruction of depth information from two-dimensional images. A new challenge is to extend these techniques to video images. Given a small set of calibrated video cameras, our goal is to render on-line dynamic scenes in real-time from new viewpoints. This paper presents an image-based rendering system using photogrametric constraints without any knowledge of the geometry of the scene. Our approach follows a plane-sweep algorithm extended by a local dynamic scoring that handles occlusions. In addition, we present an optimization of our method for stereoscopic rendering which computes the second image at low cost. Our method achieves real-time framerate on consumer graphic hardware thanks to fragment shaders.

Keywords

Image-based rendering, plane-sweep, fragment shaders.

1. INTRODUCTION

Given a set of images from different viewpoints of a scene, we set out to create new views of this scene from new viewpoints. This reconstruction problem is treated from several approaches. Some methods focus on the geometry of the scene while others use photogrametric properties. These methods can also differ on the number of input images, on the visual quality of the views created and on computation time. Most of the past work in this field concerns static scenes and tries to improve reconstruction accuracy, but past years, dynamic scene reconstruction has become a more important research area.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

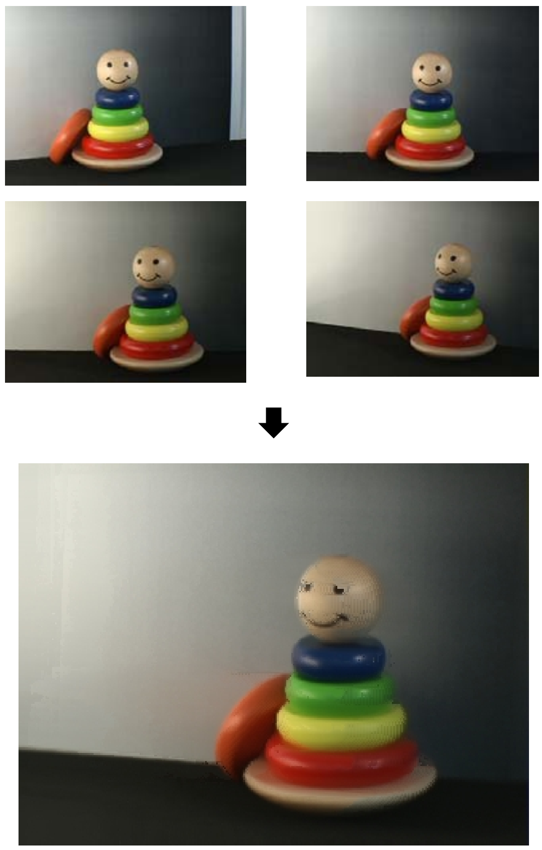


Figure 1 : A real-time reconstruction example from four cameras

In this paper, we present an overview of image-based real-time rendering for static and dynamic scenes. We detail one of these known as the plane-sweep algorithm, and present an adaptation of this method that handles occlusion. Our method achieves real-time framerate on consumer graphic hardware using fragment shaders. We also introduce an computation optimization of our method for stereoscopic rendering.

2. RELATED WORK

This section surveys previous work on real-time image-based rendering (IBR) techniques and real-time reconstruction for static and dynamic scenes.

Real-time rendering

Some image-based methods like the Plenoptic modeling [MB95], the Lumigraph [GGSC96] and the Light Field rendering [LH96] provide real-time photorealistic rendering using a large set of input 2D image samples. Nevertheless these methods require considerable off-line processing before visualization so the handling of dynamic scenes becomes very difficult. Schirmacher et al. [SLS01] extend a Lumigraph structure with per-pixel depth information using a depth-from-stereo algorithm and reach interactive-time at the cost of visual quality.

Depth-from-stereo algorithms [SS02] like SRI SVS [BBH03] provide real-time depth-maps computation from two input video streams without any special purpose hardware. However they do not provide a real-time rendering method synchronized with the real-time depth-map.

Other reconstruction methods such as texture-mapped rendering [PKV00] provide fluid navigation in the reconstructed scene but they require lengthy computation time before visualization.

Dynamic scene rendering

These last methods compute new views of a scene in real-time but most of them begin with a significant preprocessing which prevents them from computing dynamic scenes. In recent years, alternatives to this preprocessing problem and new solutions have been ardently investigated.

A first solution to this problem is to make a preprocessing on a set of videos rather than on a set of images. This allows navigation in dynamic scenes in real-time but these methods can only render playback video. Kanade et al. choose this approach with their Virtualized Reality™ System [KRN97] and achieve real-time rendering with a collection of 51 cameras mounted on a geodesic dome of 5 meters diameter. Zitnick et al. proposed a color-segmentation based stereo algorithm [ZBUWS04] providing high visual quality image in real-time from a set of 8 or more cameras but again, this method involves preprocessing.

Some other real-time techniques handle on-line video flows. Matusik et al. provide an efficient real-time rendering method with their image-based visual hulls [MBRGM00] using a set of four cameras. This method shades visual hulls from silhouette image data but therefore can not handle concave objects.

Finally, some methods like [IHA02] are based on color matching between different views, according to the epipolar constraint. Collins [C96] introduces the plane-sweep algorithm and provides basic reconstruction from binary images. Yang et al. [YWB02] extend this method to color-images and present a real-time implementation using graphic hardware. Woetzel et al. [WJKR04] adapt this method for real-time depth-mapping and introduce a first approach to handling occlusions. Geys et al. [GKV04] use a plane sweep algorithm to generate a crude depth map cleaned up using a graph-cut algorithm.

Our algorithm belongs to the latter family. We will first expose the basic plane-sweep algorithm and [YWB02, WJKR04, GKV04] contribution. Then we will detail our method.

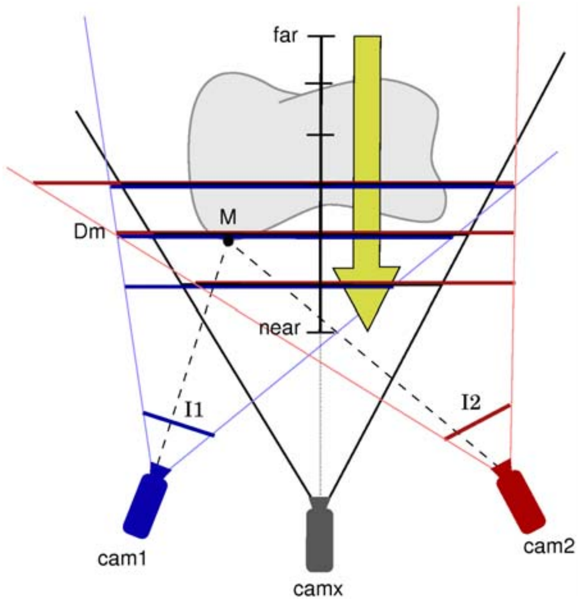


Figure 2 : Plane-sweep algorithm with two input cameras cam_1 and cam_2. M is a point of an object lying on one of the planes D_m in front of the virtual camera cam_x. The input cameras will project M's color on the same pixel of D_m.

 ISBN 1213-6972 ISBN 80-86943-09-7

3. PLANE-SWEEP ALGORITHM

The initial plane-sweep algorithm was introduced in 1996 by Collins [C96]. He first applied an edge detector filter on the input images and provided a geometric reconstruction of the scene from these binary images. The following overview is an adaptation of this method to color-images.

Overview

Given a small set of calibrated images from video cameras, we wish to generate a new view of the scene from a new viewpoint. Considering a scene where objects are exclusively diffuse, we first place the virtual camera and divide space in parallel planes D_i in front of the camera as shown in Figure 2. We project the input images onto each plane D_i in a back to front order. Let's consider a visible object of the scene lying on one of these planes at a point *M*. The input cameras will project on *M* the same color (i.e. the object color). Therefore, points on the planes D_i where projected colors match together potentially correspond to an object of the scene.

Let I_1 ... I_n denote a set of *n* calibrated images. I_x is the new image to be computed and cam_x is its virtual pinhole projective camera. We define a *near* plane and a *far* plane parallel to cam_x image plane such that all the objects of the scenes lie between *near* and *far*. For each pixel of each plane D_i, a score and a color are computed according to the matching of the colors. The plane-sweep algorithm can be explained as follows :

- initialize I_x's score
- **for** each plane D_i from *far* to *near*
  - → project all the input images $I_1 ... I_n$ on D_i as textures
  - → project D_i multi-textured on I_x
  - → **for** each pixel *p* of I_x
    - compute a score and a color according to the coherence of the colors from each camera's contribution
    - **if** the score is better than the previous ones **then** update the score and the color of *p*
- draw I_x

Figure 3 shows samples of multitextured planes D_i. When a plane pass through an object of the scene, this object becomes sharp on the multitextured image. This is the case for the wood head on the top right image.

What this method does in effect is comparing epipolar lines between the input images from each pixel of I_x. This method also provides depth-maps by drawing D_i's depth rather than a color.

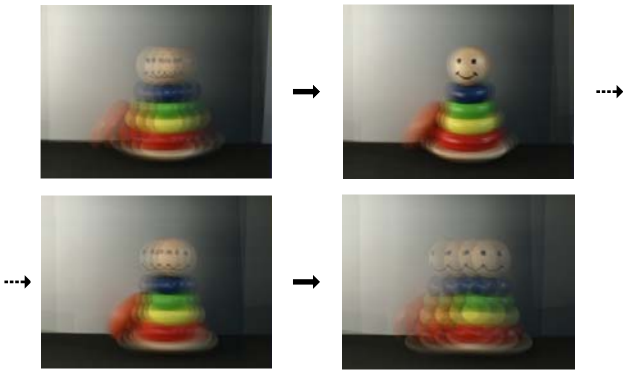


Figure 3 : Pictures associated to four planes D_i using four input images

Like several IBR techniques, this basic algorithm does not handle occlusion since the score is only computed according to the coherence of a small set of colors. We present in section 4 a modification of this algorithm that handles occlusion.

Classical score computation

Yang et al. [YWB02] propose an implementation of the plane-sweep algorithm using register combiners. For the scoring stage, they choose a reference camera cam_{base} that is closest to cam_x and compare the contribution of each input image with the reference image. Each pixel score is computed by adding the Sum of Squared Difference (SSD) from each input images. The SSD (1) compares the luminance of a pixel Y_i of an input image I_k with the corresponding luminance Y_{base} from the reference camera.

$$SSD(Y_i, Y_{base}) = \sum_i \left(Y_i - Y_{base}\right)^2 \qquad (1)$$

For more robustness, they use mipmapping to combine the pixels' score with a score computed from the same images with a lower level of detail. According to the small number of instructions, this method provides good speed results, however the input cameras have to be close to each other and the navigation of the virtual camera should lie between the viewpoints of the input cameras, otherwise the reference camera may not be representative of cam_x. Lastly, there may appear discontinuities in the computed images when the virtual camera moves and changes its reference camera. They propose a register combiners implementation and reach real-time rendering for dynamic scenes using five input cameras.

Woetzel et al. [WJKR04] propose a plane-sweep system that provides real-time depth-maps. Contrary to Yang et al. [YWB02], they do not choose a reference camera but they still compare the input images by pairs. They compute the *SSD* of each pair of input images and sort out the contribution of the two worse scores. It is a first step to handling occlusions but this method applies the same treatment to each pixel without selecting those which are concerned by occlusion and those which are not. They propose a real-time depth-map method but do not propose any rendering algorithm. This makes the comparison between our algorithms difficult.

The same problem of scoring and choosing colors among a set of colors from epipolar lines has been treated by Fitzigibbon et al. [FWZ03]. They use priors under a large set of input images to choose the color (and hence the depth) that corresponds best to most of the input images. This method is well adapted to a large set of input images and provides good results. However it requires too much computation time for real-time rendering.

Finally, Geys et al. [GKV04] propose a two steps method using two input cameras. First, a plane sweep (GPU) computes a depth map using a Sum of Absolute Differences (SAD) from the two input images. Then, an energy minimisation method (CPU) cleans up the depth map. The energy function considers temporal and spatial continuity, the previous SAD and an occlusion term derived from a background-foreground repartition of the scene elements. The energy function minimisation is solved by a graph cut method and provides a consequent improvement of the initial depth map. View dependent texture mapping of the two input images is performed to create the new view. However, this method requires a background-foreground scene decomposition with a static background. [GV05] introduces an adaptation of this method for three or more cameras.

4. OUR METHOD

We propose a new implementation which makes it possible to take into account all input images together where other methods compute images pair by pair. We introduce new methods using local strategy to compute scores allowing independent treatment of each pixel of I_x in order to handle occlusions. We also propose a new algorithm providing a stereoscopic pair of images with the second view at low cost.

New scores computation

The score computation is a crucial step in the plane-sweep algorithm. Both visual results and speedy computation depend on it. We propose a new method to compute a score according to all the input image colors instead of computing by pairs. For this purpose, we use multi-texturing functions to access each input camera color contribution.

For each pixel of I_x, we propose a finite number of positions X in the scene (one per plane D). A score is computed for each position and this score depends on the color C_i of the projections of X in each input image. We propose three methods to compute scores.

A first possibility is to set the score as the variance of each color C_i and the final color as the average of the C_i. This method is easily implemented and provides good visual results especially if the input cameras are close together. However this method does not handle occlusions. Indeed, a point viewed by all the input cameras except one will have its score and its color distorted since this camera may increase the variance and spoil the average. Nevertheless, this method implicitly treat occlusions when the virtual camera is near from an input camera which projects for each planes D_i approximatively the same image.

We also propose an iterative algorithm to reject outlier colors using a sigma clipping technique. This method first computes the variance v of the color set $S=\{C_i\}_{i=1...n}$, computes a score from v and finds the color $C_f \in S$ the furthest from the average. If this color is further than a defined distance d, then it is removed from S. This step is repeated until stability or until S contains only 2 elements. The returned score is the variance found in the last step. The choice of the constant d depends on the input cameras layout and on the scene complexity. This algorithm can be summarized as follows :

- **bool** *stable* = **false**
- $S = \{C_i\}_{i=1...n}$
- a = average(S)
- v = variance(S, a)
- *score* = scoreFunction(v, Card(S))
- **do**
  - → find the farest color $C_f \in S$ from a
  - → **if** distance(C_f, a) ≥ d **then**
    - \- $S = S - C_f$
    - \- a = average(S)
    - \- v = variance(S, a)
    - \- *score* = scoreFunction(v, Card(S))

    else *stable* = **true**

  while Card(S) ≥ 2 **and** *stable* = **false**

The *scoreFunction* weighs the variances according to *Card(S)* such that with equal variance, the set of colors with the maximum cardinal is favoured. A good score corresponds to a small variance.

Finally, we propose a third method to compute the colors' scores. This method also begins by a variance and an average computation in the color set

$S=\{C_i\}_{i=1...n}$. Then we find the color $C_f \in S$ that is the furthest from the average. A new variance and a new score are computed without this color. If this score is better than the previous one, C_f is removed from S. This step is repeated until a good score is found or until S contains only 2 elements. The score is set as the variance weighed by the cardinal of S. This algorithm can be summarized as follows :

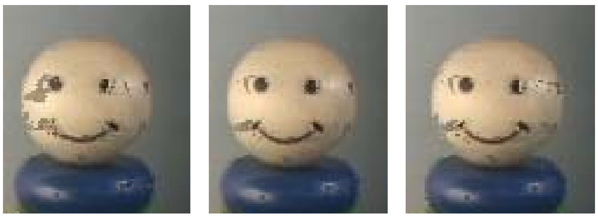
- **bool** *stable* = **false**
- $S = \{C_i\}_{i=1...n}$
- a = average(S)
- v = variance(S, a)
- *score* = scoreFunction(v, Card(S))
- **do**

  → find the farest color $C_f \in S$ from a
  → a^* = average($S - C_f$)
  → v^* = variance($S - C_f$, a^*)
  → $score^*$ = scoreFunction(v^*, Card(S)-1)
  → **if** $score^* \leq score$ **then**
  - $a = a^*$
  - $v = v^*$
  - $score = score^*$
  - $S = S - C_f$

  else *stable* = **true**

  while Card(S) ≥ 2 **and** *stable* = **false**

These three methods are easily implemented using fragment shaders. As shown in Figure 4, the two iterative methods provide better visual results, especially when the input camera are placed in a 1D arc configuration which increase the occlusions effects.

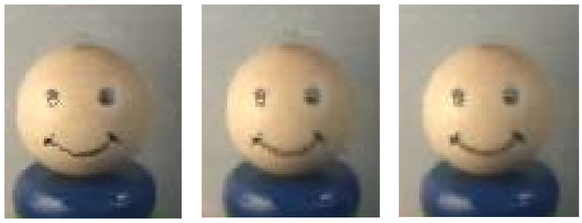


(a) (b) (c)

Figure 4: image (a) is computed using the variance and the average, (b) using the sigma clipping technique and (c) using the second iterative method.

Neighborhood with mipmapping

For more robustness during the scoring stage, we take into account the neighborhood color contribution of each pixel. Mipmapping provides access to the same image but with a lower level of details (lod) and hence provides the average color of the neighborhood of the current pixel. For each pixel score, we combine the score computed using different lods. Yang et al. [YWB02] propose a summation over a box-filtered lod pyramid but only one additional mipmap level works well with our method and more mipmap levels do not improve the visual results. This is illustrated in Figure 5.



(a) (b) (c)

Figure 5: Images computed with different mipmap levels : (a) no additional mipmap level, (b) 1 mipmap level and (c) 2 mipmap levels.

Stereoscopic rendering

Virtual reality applications often requires stereoscopic display to increase immersion and most of these applications have to render the scene twice. But a lot of information such as diffuse lighting for example can be shared for both views. Concerning IBR techniques, depth-mapping is often view-dependant and hence the two new views must be computed separately. The plane-sweep algorithm computes local score associated to scene points. This information can be shared for several virtual cameras. We extend our method with a low cost algorithm providing the second view.

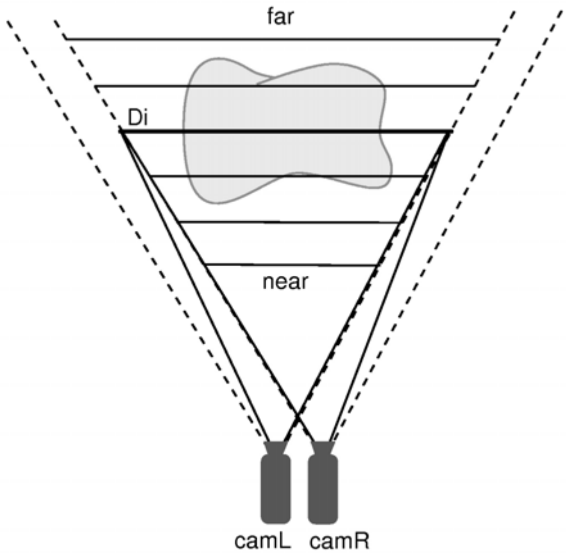


Figure 6 : Each plane D_i is common to the two views, but their projection differs

Stereoscopic rendering must satisfy several conditions concerning virtual camera parameters

[SC97]. In particular, both cameras must have their principal ray parallel to avoid vertical parallax in the stereoscopic image. Let cam_L and cam_R be a pair of virtual cameras satisfying this constraint and $D_{1...m}$ a set of planes parallel to these cameras' image plane. As shown in Figure 6, the score and the color computation of a plane D_i is common for both cam_L and cam_R. Only the projection of D_i on the two cameras will differ. The score computation is a central task in the plane-sweep algorithm, so sharing this stage among the two views provides a consequent gain in computation time. Thus, our plane-sweep method must be modified as follows :

- initialize I_L and I_R's score
- **for** each plane D_i from *far* to *near*
  - → project all the input images $I_1...I_n$ on D_i
  - → render D_i on two textures *texScore* and *texColor* : **for** each pixel of I_{tmp}
    - - compute a score and a color according to the coherence of the colors from each camera's contribution
  - → copy *texScore* and *texColor* on D_i
  - → project D_i multi-textured on I_L and I_R
  - → **for** each pixel of I_L and I_R
    - - **if** the score is better than the previous one **then** update the score and the color
- draw I_L and I_R

For each planes D_i, this method first computes scores and colors and stores them in two textures. In a second pass, these two textures are copied on D_i and projected on the two virtual cameras. The first pass requires off-screen rendering performed by Frame Buffer Objects (FBO) and Multiple Render Target (MRT). This step can also be achieved using p-buffers with a small frame rate penalty.

Thus, this method can easily be implemented such that all the image data stay in the graphic card and hence avoid expensive data transfers between the graphic card and the main memory.

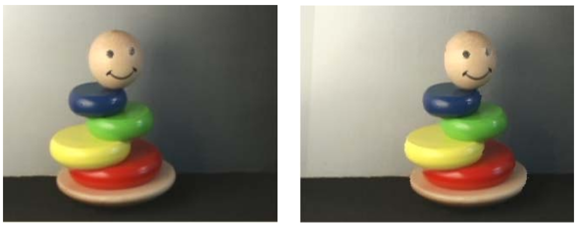


Figure 7: Real-time stereoscopic pair (cross vision)

Figure 7 shows a stereoscopic pair rendered in real-time. Note that the fusion of the two images decreases the imperfection impact of the images.

As illustrated in Table 1, stereoscopic rendering achieves a 15% frame rate decrease instead of the 50% expected by rendering twice the scene.

Implementation

Input cameras are calibrated using the *gold standard algorithm* [HZ04]. We implemented our method on OpenGL 2.0 and we use OpenGL Shading Language for the scoring stage.

For more accuracy, the texture coordinates are computed using projected textures directly from the camera projection matrices. We use multitexturing in order to get access to each texture during the scoring stage. Each score is computed with fragment shaders using mipmapping. They are stored in the `gl_FragDepth` and the colors in the `gl_FragColor`. Hence we let OpenGL select best scores with the z-test and update the color in the frame buffer.

To compute a depth-map rather than a new view, we just set the `gl_FragColor` to the `gl_FragCoord.z` value.

Most of the work is done by the graphic card and the CPU is free for others tasks.

5. RESULTS

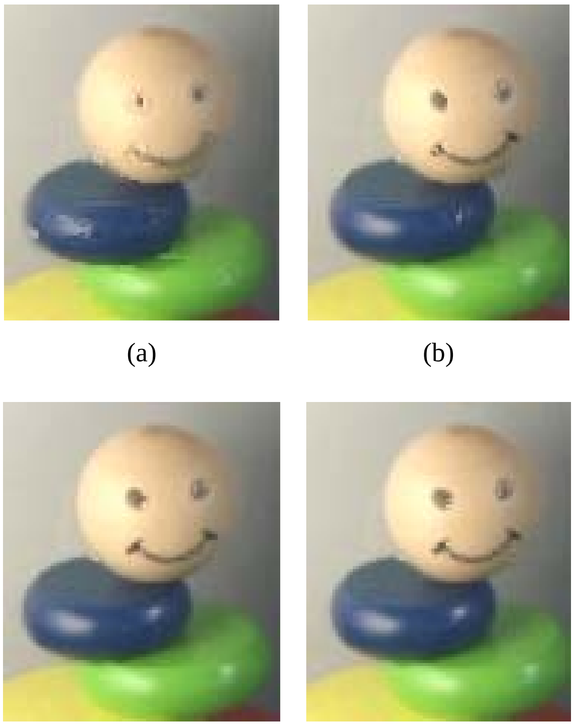
We tested our methods on an Athlon AMD 1GHz with a Nvidia GeForce 6800GT. We used four tri-CCD Sony DCR-PC1000E cameras for the input images acquisition. The white balance is essential in a plane-sweep algorithm. Indeed, we must homogenize the camera color range such that any point in the scene is seen with the same color from each camera. For our tests, we used the manual white balance provided by the tri-CCD cameras but for more accuracy, we planed to use a color calibration method as proposed by Magnor [M05].

Table 1 shows the framerate we obtain with 4 input cameras.

Number of plans *D*	Simple variance	Method 1 and 2		Stereo-scopic
10	140	85	91	110
30	43	28	30	38
50	30	17	18	25
100	15	9	9	13

Table 1. Frame rate in frame per second for a 320x240 image from 4 input cameras.

 ISBN 1213-6972 ISBN 80-86943-09-7



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

Figure 8: Number of planes used for each scene : (a) 5 planes, (b) 10 planes, (c) 30 planes and (d) 50 planes

The computation time depends on the number of planes we choose to discretize the scene. Our tests indicate that after 50 planes, the quality difference becomes neglible (Figure 8).

We are presently working on examples of on-line dynamic scenes.

6. CONCLUSION

This paper presents a plane-sweep method that allows real-time rendering of on-line dynamic scenes. Except for *near* and *far* planes, it does not require any prior knowledge of the scene. This method can be implemented on every consumer graphic harware that supports fragment shaders and therefore frees CPU for other tasks. Furthermore, our scoring method enhances robustness and implies fewer constraints on the position of the virtual camera, *i.e.* it does not need to lie between the input camera's area.

We propose to extend our research in optimisation of D_i planes repartition in order to reduce its amount without depreciating the visual result. We also intend to achieve a better stereo viewing result by producing pairs of virtual cameras with non symetric projection pyramid in order to save space on the edges of the stereo images [GPS94].

7. REFERENCES

[BBH03] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. *Advances in Computational Stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 993-1008, 2003.

[C96] Robert T. Collins, *A Space-Sweep Approach to True Multi-Image Matching*, in proc. Computer Vision and Pattern Recognition Conf., pages358-363, 1996.

[FWZ03] Andrew Fitzgibbon, Yonatan Wexler and Andrew Zisserman, *Image-based rendering using image-based priors*, 9th IEEE International Conference on Computer Vision (ICCV 2003), pages 1176-1183, 2003.

[GGSC96] J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen, *The lumigraph*, SIGGRAPH, pages 43-54, 1996.

[GKV04] Indra Geys, T. P. Koninckx and L. Van Gool, *Fast Interpolated Cameras by combining a GPU based Plane Sweep with a Max-Flow Regularisation Algorithm*, in proc. of second international symposium on 3D Data Processing, Visualization & Transmission - 3DPVT'04, pages 534-541, 2004.

[GPS94] V.S. Grinberg, G. Podnar and M. Siegel, *Geometry of Binocular Imaging*, Stereoscopic Displays and Virtual Reality Systems, Vol. 2177, pages 56-65, 1994.

[GV05] Indra Geys and L.Van Gool, *Extended view interpolation by parallel use of the GPU and the CPU*, in proc. of IS&T SPIE, 17th annual symposium on electronic imaging - videometrics VIII, vol. 5665, pages 96-107, 2005.

[HZ04] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, second edition, Cambridge University Press, ISBN: 052154051, 2004.

[IHA02] M. Irani, T. Hassner and P. Anandan. *"What does the scene look like from a scene point?"*, Proc. ECCV, pages 883-897, 2002.

[KRN97] Takeo Kanade, Peter Rander and P. J. Narayanan, *Virtualized Reality: Constructing Virtual Worlds from Real Scenes*, IEEE MultiMedia, volume 4, pages 34-47, 1997.

[LH96] Marc Levoy and Pat Hanrahan, *Light Field Rendering*, SIGGRAPH, pages 31-42, 1996.

[M05] Marcus A. Magnor, *Video-Based Rendering*, Editor : A K Peters Ltd, ISBN : 1568812442, 2005.

[MB95] Leonard McMillan and Gary Bishop, *Plenoptic Modeling: An Image-Based Rendering System*, SIGGRAPH, pages 39-46, 1995.

[MBRGM00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler and Leonard McMillan, *Image-Based Visual Hulls*, in proc ACM SIGGRAPH, pages 369-374, 2000.

[PKV00] M. Pollefeys, R. Koch, M. Vergauwen and L. Van Gool, *Automated reconstruction of 3D scenes from sequences of images*, ISPRS Journal Of Photogrammetry And Remote Sensing (55)4, pages 251-267, 2000.

[SC97] StereoGraphics Corporation, *Developer's Handbook : background on creating images for CrystalEyes and SimulEyes*, StereoGraphics Corporation, 1997.

[SLS01] Hartmut Schirmacher, Ming Li and Hans-Peter Seidel, *On-the-fly Processing of Generalized Lumigraphs*, Proc. EUROGRAPHICS 2001, Eurographics Association, pages 165-173, 2001.

[SS02] Daniel Scharstein and Richard Szeliski,*A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*, IJCV, 47, pages 7-42, 2002.

[WJKR04] Woetzel, Jan, Koch and Reinhard, *Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware*, in proc. of 17th International Conference on Pattern Recognition (ICPR 2004), pages 741-744, 2004.

[YWB02] Ruigang Yang, Greg Welch and Gary Bishop, *Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware*, in proc. of Pacific Graphics, pages 225-234, 2002.

[ZBUWS04] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder and Richard Szeliski, *High-quality video view interpolation using a layered representation*, in proc. ACM SIGGRAPH, pages 600-608, august 2004.

View-dependent Tetrahedral Meshing and Rendering using Arbitrary Segments

Ralf Sondershaus
WSI / GRIS
University of Tübingen, Germany
sondershaus@gris.uni-tuebingen.de

Wolfgang Straßer
WSI / GRIS
University of Tübingen, Germany
strasser@gris.uni-tuebingen.de

ABSTRACT

We present a meshing and rendering framework for tetrahedral meshes that constructs a multi-resolution representation and uses this representation to adapt the mesh to rendering parameters. The mesh is partitioned into several segments which are simplified independently. A multi-resolution representation is constructed by merging simplified segments and again simplifying the merged segments. We end up with a (binary) hierarchy of segments whose parent nodes are the simplified versions of their children nodes. We show how the segments of arbitrary levels can be connected efficiently such that the mesh can be adapted fast to rendering parameters at run time. This hierarchy is stored on disc and segments are swapped into the main memory as needed. Our algorithm ensures that the adapted mesh can always be treated like a not-segmented mesh from outside and thus can be used by any renderer. We demonstrate a segmentation technnique that is based on an octree although the multi-resolution representation itself does not rely on any paticular segmentation technique.

Keywords: multi-resolution meshes, tetrahedral meshes, view-dependent rendering, volume rendering

1 INTRODUCTION

Tetrahedral meshes are often used as finite element meshes that discretize a volumetric domain for scientific simulations like computational fluid dynamics (CFD). Modern simulation environments typically use meshes that contain millions of tetrahedra.

The simulations carry data along with a tetrahedral mesh. The data values are usually scalar values like temperature or pressure, or vector values like velocity, and can be attached to the vertices, edges, border faces or to the tetrahedra.

The emerging need to visualize the simulation data has introduced tetrahedral meshes to volume visualization. Modern algorithms render up to about one million of projected tetrahedra per second [KQE04] or can extract isosurfaces of about two millions of tetrahedra per second [KSE04]. This is not enough to render a model like the F16 interactively (figure 8) that contains about 6 million tetrahedra.

In this paper, we present an *out-of-core data structure* that enables such a big mesh to be simplified with a small memory footprint. The data structure is built on segments which are swapped efficiently to and from the core memory as needed. Additionally, we introduce a *multi-resolution framework* which is built on a hierarchy of segments and can be used for *interactive volume rendering*.

Our out-of-core data structure partitions the mesh into segments by merging leaves of an octree. The segmentation adapts to the details of the mesh and thereby creates segments which contain similiar number of vertices (and tetrahedra). Subsequent algorithms (like simplifiers) process the mesh one segment after the other and need small memory footprints. As a by-product, the vertices are reordered as they are assigned to the segments which results in a better performance of a subsequent algorithm because it tends to reduce cache-misses.

Based on the out-of-core data structure, we introduce a multi-resolution framework. Many previously published multi-resolution frameworks work with a hierarchy of vertices and decide per frame which vertex is to be split or which edge is to be collapsed in order to refine or coarsen the mesh. Because a huge mesh can contain millions of points, the adaptation of the mesh can be become time-consuming even if priority queues are used. We construct a hierarchy of segments where segments of arbitrary resolution levels can connect to each other (however, they need to share the same border vertices). The hierarchy of segments is used at run-time to adjust the mesh to viewing parameters and performs better than a vertex hierarchy because fewer nodes must be tested to be refined or coarsened and no dependencies between nodes are needed.

For triangle meshes, the big issue is to connect different segments correctly if the segments belong to different resolution levels. Many previous algorithms store dependencies between segments in a directed acyclic graph or restrict the resolution levels to differ by at

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.

most one between adjacent segments in order to ensure correct connections between segments. Our model can connect arbitrary resolution levels by fixing the borders between segments.

For tetrahedral meshes, the adjacency information for every tetrahedron must additionally be set in order for subsequent algorithms (like MPVO sorting in volume rendering) to work correctly. Our multi-resolution framework introduces a so-called 0-segment which handles the adaptation of the adjacency information efficiently. The volume renderer always sees just one consistent mesh and can run its sorting and rendering routines as if no multi-resolution mesh is used.

Our multi-resolution model does not depend directly on how the mesh is segmented but can work on any segmentation. Instead of the octree-based structure, other techniques like vertex clustering could be used. Furthermore, we do not rely on edge collapse based simplifications but could use any simplification technique.

2 RELATED WORK

Simplification, multi-resolution, and out-of-core techniques are described shortly because they are important to our technique. A small overview to volume rendering techniques is given.

Simplification. We restrict our overview to approaches that are based on edge collapses. For other simplification techniques we refer to mesh decimation [RO96] or TetFusion [CM02].

Popovic et al. [PH97] have extended the Progressive Mesh approach [Hop96] to general simplicial complexes but do not take into account how the underlying scalar field of a tetrahedral mesh is approximated.

Staadt et al. [SG98] have applied the Progressive Mesh approach to tetrahedral meshes. The edge collapses are sorted by an error heap that uses a cost function which considers various errors like scalar field error or volume and shape deformation.

Cignoni et al. [CCM$^+$00] have characterized the field and domain errors of an edge collapse and present various techniques to predict these errors reliably. The field error is introduced by approximating the original scalar field of the mesh whereas the domain error is introduced by reducing the boundary of the mesh.

Kraus et al. [KE00] have simplified non-convex meshes and can change the topology of the mesh during simplification. Chiang [CL03] have preserved the topological structure of isosurfaces of the mesh during simplification.

Garland [GZ05] extended the quadric error metrics to arbitrary simplices (and to tetrahedra in particular) and showed that this approach produces high-quality approximations that automatically take domain and field errors into account.

Out-of-Core Data Structures. Cignoni et al. [CMRS03] use an octree to partition a large tetrahedral (or triangle) mesh into segments. Each segment can be modified independently of the other segments. Simplification algorithms are adapted to process the mesh on a per-segment basis.

Gumhold et al. [GI03] construct a segmentation of a huge triangle mesh by sorting the vertices into a regular grid and merging grid cells into segments that contain approximatively the same number of vertices. Triangles are sorted into the segments according to their center points.

Isenburg et al. [IL05] introduced streaming meshes which are defined as a new file format that interleaves triangle and vertex definitions and does not introduce a vertex until it is indexed by a triangle. Furthermore, it is marked if no subsequent triangle indexes a vertex anymore such that this vertex can be safely deleted. Streaming meshes are highly efficient to both mesh compression and mesh simplification [VCL$^+$05].

Multi-resolution representations. Many multi-resolution representations [CMRS03, DDFM$^+$04, CDFL$^+$04] construct a binary vertex hierarchy by edge collapses. At run-time, a front through the hierarchy defines a valid mesh. Vertices on the front can be split (which refines the mesh) or collapsed (which coarsens the mesh). In order for the mesh to be valid, not all splits or collapses are valid. Geometric and topological conditions need to be checked and the operation is allowed only if the conditions are fulfilled. These conditions can be enforced by performing additional operations which results in additional costs.

Cignoni et al. [CGG$^+$04] use longest-edge bisection of a tetrahedral mesh in order to decompose the spatial domain of a huge polygonal mesh and to construct a hierarchical decomposition of this polygonal mesh. The mesh segments that are contained in a tetrahedra diamond are simplified leaving the border vertices unchanged, i.e. those vertices that connect different segments. Using longest-edge bisection results in a hierarchy which ensures that neighboring segments can differ in at most one resolution level.

Volume Visualization. A standard technique for direct volume rendering of unstuctured tetrahedral meshes is the projected tetrahedra algorithm of [ST90] that has been greatly enhanced by the pre-integration technique of [MHC90, RKE00].

The tetrahedra are sorted from back to front which can be done for all acyclic tetrahedral meshes and because a tetrahedron is always convex. The tetrahedra are projected onto the view plane and decomposed into 1 - 4 triangles according to the positions of the projected vertices. These triangles are rendered using alpha blending from back to front.

3 OUT-OF-CORE DATA STRUCTURE

We design a data structure that is suited to handle large tetrahedral meshes memory efficiently as well as to support volume rendering at run-time. Therefore, the mesh is partitioned into segments such that the segments are stored on disc, can be loaded independently to main memory and can be written back to disc.

Every segment contains a number of vertices (such that a vertex of the original mesh belongs to exactly one segment) and a number of tetrahedra (a tetrahedra of the original mesh belongs to exactly one segment).

The index of a vertex consists of an index-pair (s_i, l_i) with a segment index s_i and a local index l_i which specifies the vertex within the segment s_i. We encode this index-pair as a bit field of 32 bits. A tetrahedron is also addressed as an index-pair with l_i as the local index of the tetrahedron.

Because all vertices and tetrahedra need to be addressable with this address space, the number of vertices and tetrahedra within a segment should be balanced.

The multi-resolution model (section 5) adds new segments to the mesh that are simplified versions of the original segments. Every simplified segment has an error associated with it such that the volume renderer can compute which segment is to be swapped in and out from the main memory. In order to achieve a good error estimation for each segment, we need segments that contain vertices with similiar attribute values and tetrahedra with similiar sizes.

Using these objectives as a starting point, we construct the segmentation as follows.

3.1 Construction

Although we could use the techiques of [CMRS03] or [GI03], we found both only partly applicable to our models. An octree partitions the vertices of a mesh fast and robust. Its leaves reflect the density of the points in the mesh which can be significantly different in different areas of a tetrahedral mesh (for an example see figure 8, left). But the number of vertices that are sorted into cells can differ highly such that some cells contain almost no vertices whereas other contain many. This leads to segments of different sizes which can result in memory and address space defraction.

A grid has the disadvantage that the user must specify its resolution and that dependending on the resolution the variation of the number of vertices of the grid cells differ highly. But it is easy to be implemented out-of-core. It is mandatory to combine cells afterwards to form segments in order to obtain balanced segments. For tetrahedral meshes, the size of the grid cells needs to be very small in order to catch the fine-detailed areas of the mesh which results in a huge number of cells that contain data.

We combine both approaches. First, an octree is constructed for the vertices of the mesh. Afterwards, the leaves of the octree are considered as nodes in a (undirected) graph. The edges of the graph reflect the neighborhood of the leaves but contain edges between cells only that share a face and have similiar sizes (we restrict the size difference to be at most two). This ensures that areas of the mesh with a similiar point density are merged into one segment because the size of an octree leaf reflects the point density of the mesh.

A graph partitioning algorithm (we use Metis 4.0) is called which constructs a partition of the graph. Every partition forms now a segment and consists of a collection of leaves of the octree. Every leaf belongs to exactly one segment.

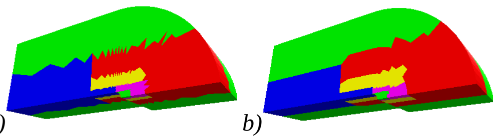


Figure 1: *Segmentations that sort the tetrahedra (a) according to their center vertices into the segmentation, or (b) according to the smallest segment index of their four vertices.*

Afterwards, we sort the tetrahedra into the segments as follows. The four vertices of a tetrahedron are assigned to their segments. The tetrahedron is assigned to the segment of the vertex with the smallest segment index. We do not need to compute the center of the segment (as it is done by [CMRS03] and [GI03]) and found this method to produce well-shaped borders that are sufficient for our purposes, see figures 1 and 2.

The user specifies the average number of vertices that are to be stored in each segment. The octree is constructed to contain at most this number of vertices in its leaves. The graph matching combines leaves into segments such that a balanced segmentation is achieved.

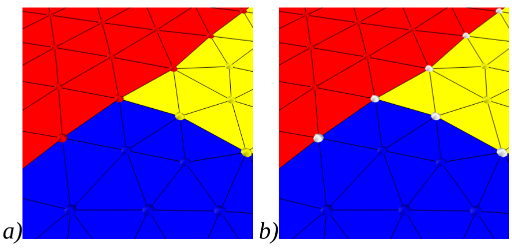


Figure 2: *(a) A tetrahedron is sorted to the segment of its index with the smallest segment index. Red is smaller than yellow which is smaller than blue. Note that a tetrahedron may reference vertices from other segments. (b) The white vertices on the boundary belong to the* 0*-segment as described in section 5.*

Because subsequent algorithms process the mesh by traversing the segments one after another, the segments need to be stored in an order that neighboring segments are traversed together. Therefore, we sort the segments by their minimal points, first in x-, followed by y- and by z-coordinates. More elaborated techniques could be applied here.

The segments are stored in a single file on disc in the order of the sortation. For every segment, we store

1. The geometry of the vertices using the local ordering of the vertices within the segment.
2. The attributes of the vertices (if any).
3. For every tetrahedron its four vertex index-pairs.
4. The attributes of the tetrahedra (if any).
5. For every tetrahedron its four adjacent tetrahedra.
6. The indices of all segments that are incident to this segment (i.e. share at least one vertex).

In order to traverse the tetrahedra of the mesh, we store adjacency indices (number 5 above). Every tetrahedron t stores one index-triple (s_i, t_i, c_i) for each of its vertex index-pairs that points to the tetrahedron that is opposite to the vertex. The index-triple encodes the segment index s_i, the local index t_i (within the segment s_i) and a code $c_i \in \{0,1,2,3\}$. The code specifies the vertex inside the adjacent tetrahedron that is opposite to the shared face.

The interface of the data structures allows for loading a particular segment as well as to request a single vertex or tetrahedron such that the segment that this vertex belongs to is automatically loaded. A Last-Recently-Used queue keeps track of all loaded segments and stores segments that have been changed back to disc if they are not needed any more.

4 SIMPLIFICATION

Using our data structure, the simplifier traverses the mesh one segment after the other and simplifies each segment. Our simplifier uses edge collapses and is steered by a priority queue that sorts all possible collapses of a segment by the error that they introduce. The error is evaluated by using quadric error metrics of Garland et al. [GZ05].

A quadric error metric measures the squared geometric and attribute distances of points to hyperplanes that are spanned by the (original) tetrahedra. A special penalty error can be introduced at boundary vertices in order to preserve the boundary of the mesh well.

The vertices $\mathbf{v} = (v_x, v_y, v_z)$ of the mesh and their attributes $f_{i,v}$ are embedded into a $3+k$ dimensional space with $\mathbf{p} = p(v_x, v_y, v_z, f_{1,v}, ..., f_{k,v})$. For every tetrahedron $t = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$, a local coordinate system $\mathbf{e}_0$, $\mathbf{e}_1$, $\mathbf{e}_2$ is constructed by orthonormalizing the vectors $\mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{p}_2 - \mathbf{p}_0$, and $\mathbf{p}_3 - \mathbf{p}_0$.

Then, the quadric error function Q of a tetrahedron can be written with a symmetric, positive semi-definite matrix $\mathbf{A}$, a vector $\mathbf{b}$, and a scalar value c as

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} - 2\mathbf{b}^T \mathbf{x} + c$$

with

$$\mathbf{A} = \mathbf{I} - \sum_{i=0}^{2} \mathbf{e}_i \mathbf{e}_i^T \qquad \mathbf{b} = \mathbf{A}\mathbf{p} \qquad c = \mathbf{p}^T \mathbf{A} \mathbf{p}$$

where $\mathbf{p}$ is the barycenter of the tetrahedron.

The quadric error of a vertex approximates the sum of the squared distances to its incident hyperplanes and can be computed by summing all matrices $\mathbf{A}$ component-wise as well as all vectors $\mathbf{b}$ and all scalars c of the incident tetrahedra.

For an edge collapse, the metrics (i.e. $\mathbf{A}$, $\mathbf{b}$, and c) of both collapsing vertices are summed component-wise. The point that minimizes this quadratic error function can be found by solving the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. Because $\mathbf{A}$ is symmetric and positive semi-definite, a cg solver can be used that takes the middle point of the collapsing edge as starting point.

5 MULTI-RESOLUTION MODEL

The data structure and the simplifier presented so far can be used to simplify a tetrahedral mesh as a whole. But for view-dependent rendering the mesh needs to be adapted to viewing parameters and thus different simplification (or resolution) levels need to be merged into one mesh at run-time.

Therefore, we build a hierarchy of segments where each segment can connect to its neighbors regardless of their resolution levels. We first describe how the hierarchy is constructed and show after that how the adjacency information between segments can be updated.

5.1 Construction

The segmented tetrahedral mesh is stored on disc as described above. We create new segments to build up a hierarchy as follows and append the new segments at the end of the file. Each segment stores all the information described in section 3.

First, we copy each segment of the original mesh and simplify the copies (as described above) independently. The vertices that are referenced by different segments remain unchanged (i.e. the vertices on the boundary between at least two segments). We end up with a coarse approximation of each segment where the segments are still connected at the original resolution level. The simplified copies are inserted as parents of their original segments into the hierarchy, see figure 6.

Secondly, we construct an undirected graph whose nodes are the (simplified) segments and whose edges are between every pair of (simplified) segments that share at least one tetrahedral face. The nodes are weighted by the number of vertices in their segments. We find a matching of the graph using a greedy algorithm and end up with pairs of nodes (segments).

Each pair is merged into a new segment and the new segment is simplified. The vertices that have connected

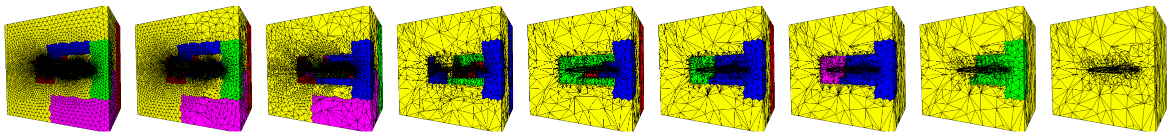


Figure 3: The construction process of the NASA fighter dataset merges two segments into a new segment which is simplified. The colors of the segments can change from picture to picture.

both segments belong to the new segment now and are simplified. The vertices that are on the boundaries between the new segments remain unchanged as shown in figure 4.

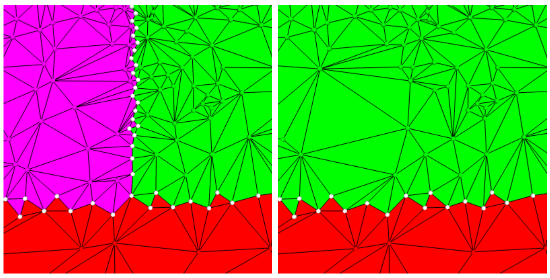


***Figure 4:** The purple and the green segment are merged and simplified. The shared vertices are removed but the vertices to the red segment are left unchanged (white vertices).*

Iterating this strategy leads to the algorithm in figure 5. Every iteration constructs the neighboring graph and computes pairs of segments. Both segments of a pair are merged into a new segment which is simplified. The hierarchy is constructed by building a (at most levels binary) tree where the new segment is the parent of both merged segments as shown in figure 6.

Because the border vertices between segments are left unchanged, different resolution levels automatically connect to each other. Figure 3 shows the construction process for the NASA fighter dataset.

```
For every segments s
    s_new = copy segment s;
    add s_new as parent of s;
    simplify s_new;
While the number of roots > 1
    construct the node graph G;
    find pairwise matching M of G;
    For every pair (s1,s2) in M
        s_new = merge segments s1 and s2;
        add s_new as parent of s1 and s2;
        simplify s_new;
```

***Figure 5:** Every iteration finds pairs of segments that are merged into a new segment and simplified.*

The final mesh segments are written in breadth-first order to disc together with the hierarchy. Using a breadth-first order enhances file accesses because neighboring segments are likely to be stored near together. Each segment is stored as described in section 3.

5.2 Usage

Refining one segment means to replace the mesh of the segment by the meshes of its children. Both children can now be refined independently of each other. For instance, the first child could be refined again and again while the second child remains unchanged. This would lead to a high resolution level that automatically connects to the low resolution level of the second child (but with the limitation that both the high and the low resolution segments need to share the same vertices on their border).

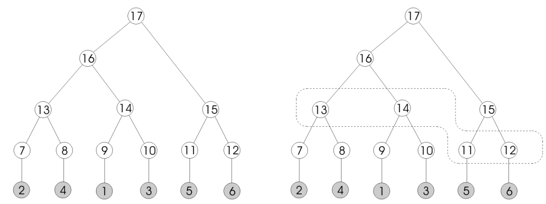


***Figure 6:** The segment hierarchy stores how the segments are merged into their parent segments (left). The segments of the original mesh are gray. A front through the hierarchy corresponds to a valid mesh (right).*

Coarsening a segment means to replace the segment itself as well as its sibbling (in the binary hierarchy) by the coarser mesh of the parent.

A valid mesh is defined as a list of segments within the hierarchy that has exactly one segment in every path from the root down to the leaves as shown in figure 6 (right). We call this list a segment front. This corresponds to the well-known vertex front in vertex-based multi-resolution models [DDFM$^+$04].

5.3 The 0-segment

The meshes of the segment front form a valid tetrahedral mesh. During rendering, the adjacencies between all tetrahedra are often needed (for instance for MPVO-based sorting [Wil92]). A fast method to adapt the adjacencies between segments is needed whenever a segment is replaced by another segment. We introduce a special segment that handles the adjacency between any two neighboring segments. Because this special segment has the unique segment index 0, we call it 0-segment.

The 0-segment is defined as the cut of all segments of the original mesh. Thus, it contains

- all vertices that are on boundaries between two neighboring segments in the original mesh (white vertices in figure 1b), and
- all triangles that form the border between two neighboring segments, and
- (theoretically) all edges (not stored).

The triangles of the 0-segment are now used as a buffer (or a docking station) between two segments. Instead of storing the index of the adjacent tetrahedron, the index of the shared triangle in the 0-segment is stored in the adjacency information. Because the triangles never change and have always the same index, the adjacency to this triangle can be stored in the mesh (and in the file).

Each triangle stores two adjacency index-triples that point to tetrahedra in the adjacent segments, see also figure 7b. The tetrahedra of a segment that have a border to the 0-segment store an index-triple (s_i, t_i, c_i) as usual that points into the 0-segment with $s_0 = 0$ and t_i as the triangle index. c_i can be 0 or 1 and points to one of the two adjacency index-triples of the triangle.

When a segment s replaces another segment r, it must update the 0-segment as follows. All border tetrahedra of s are traversed. At least one of the four adjacency index-triples of these tetrahedra point to a triangle t in the 0-segment (the other index-triples point to tetrahedra inside s itself). The index-triple of t points to the segment r and must be replaced by the index-triple of s, i.e. (s_i, t_i, c_i) where $s_i = s$, t_i is the index of the border tetrahedron and c_i is the local index of the opposite vertex within t_i, see also figure 7.

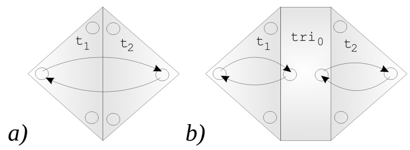


***Figure 7:** Instead of storing the index-triple of the neighboring tetrahedron at the border of two segments (a), the index of the triangle in the* 0*-segment is stored (b).*

In order to find all border tetrahedra fast, they are stored before all other (inner) tetrahedra in the file. So iterating all border tetrahedra can be accomplished by iterating over all tetrahedra until a non-border tetrahedron is found.

The 0-segment does not need to exist at the construction phase. It must exist only at run-time when segments are to be exchanged very fast and can be computed once after (or before) the construction phase.

6 VIEW-DEPENDENT RENDERING

In order to adapt the mesh to current viewing and classification parameters, we need to decide which segments of the segment front must be coarsened (replaced by the parent) or refined (replaced by the children). Therefore, the following values are stored with each segment

- The histogram H of the attribute values (which is a lookup-table of resolution N with normalized entries $H_i \in [0,1]$), and
- A look-up table E of the same resolution N which specifies the maximal error that the segment contains for the according attribute value, and
- The (axis aligned) bounding box.

The user can specify a maximal field error ε_{max}.

For each frame, the segment front is traversed. Every segment of the front is marked by the tags COARSEN, REFINE, and NOTHING that help later to adapt the mesh:

1. If the bounding box is outside the view-frustum and if the sibbling exists, mark as COARSEN, else
2. Compute the average $S = \sum_i^N H_i E_i \alpha_i$ where the sum runs over all histogram values, H_i is the (normalized) i-th histogram value, E_i is the according error and α_i is the according classified opacity.

   If $S > \varepsilon_{max}$, mark as REFINE, else
3. Mark as NOTHING.

After all segments of the front are marked, the front is traversed again and every segment is adapted:

1. If a segment is marked as REFINE, it is replaced by its children.
2. If a segment and its sibbling are marked as COARSEN, both are replaced by their parent.

The histogram H as well as the look-up table E are computed during the construction of the multi-resolution model. Every edge collapse introduces a particular error for a scalar field attribute which is stored in the look-up table E if it is greater than the already stored error.

A simple LRU queue keeps track of the mapped segments. We refer the reader to the more elaborated caching methods of for instance [YLPM05].

7 RESULTS

We implemented the technique with memory mapped files which are a operating system opportunity for memory allocation such that parts of a file can be directly mapped into memory. The operating system performs all necessary swapping.

Name	# Vertices	# Tetra	# Segments	Min Vertices per segment	Max Vertices per segment	Min Tets per segment	Max Tets per segment	Time hh:mm:ss
Seaway	102,165	524,640	18	3,050	5,989	17,382	38,929	0:00:10
Fighter	256,614	1,403,504	48	3,175	5,078	21,648	31,660	0:00:15
Rbl	730,273	3,886,728	131	2,714	6,599	17,378	36,284	0:00:26
F16	1,124,648	6,345,709	212	2,874	6,169	26,104	37,087	0:00:58

Table 1: *The properties of the datasets and the timings for the construction of the segmented mesh from the original mesh. The octree was steered to contain at most 5,000 vertices per leaf node.*

So if a segment s replaces another segment, our implementation calls the operating system to map the parts of the file that correspond to s into the memory. We found that the operating system (we use Windows XP) needs nearly constant time to map a segment if the mesh part has been accessed some time before due to caching. However, sometimes the caching misses and delays of at most one second may occur.

The frame rates depend mainly on the power of the volume renderer that uses preintegrated projected tetrahedra. Because the size of the segment front is small, the costs for checking if a segment can be refined or coarsened, are neglectable. We experienced frame rates of about 3-4 frames per second with a workload of about 250,000 tetrahedra. We can render the F16 model interactively which is impossible for the full resolution mesh (it would take at least 6 seconds to render a single frame).

In average, 80% of the time of a frame is used by the volume renderer whereas 16% are used for reloading segments (file IO) and 4% are used for segment adjacency adaption (measured average values for the NASA Fighter dataset, the other datasets perform similiar).

The construction timings of the segmented meshes (section 3) are shown in table 1. Most of the time is needed for file IO. For the Rbl dataset, for instance, the IO to write the mesh to disc needed 20 seconds (out of the 26 seconds total construction time).

The main time was spent to simplify the meshes and to construct the hierarchy, see table 2. Although the timings do not compare to the (much faster) timings of Lindstrom [VCL+05], we differ from Lindstrom because we need to store all segments and do not use the randomized edge collapses.

Furthermore, the file sizes are huge because we store each segment in a raw format such that it is ready to be mapped to memory.

Name	# Tetra in base mesh	Time hh:mm:ss	file size [MB]
Seaway	18,763	0:13:47	67
Fighter	30,042	0:19:23	101
Rbl	259,363	0:31:05	349
F16	84,246	1:08:23	514

Table 2: *The simplification timings and the sizes of the stored multi-resolution files.*

8 CONCLUSION AND FUTURE WORK

We presented an out-of-core data structure that enables to simplify a tetrahedral mesh with a small memory footprint. The segments are constructed by combining leaves of an octree such that the segments contain similiar parts of the mesh.

A multi-resolution hierarchy is built based on the segments where pairs of segments are merged and simplified. The segments connect to each other using the 0-segment. It allows for the multi-resolution mesh to adapt its adjacency information efficiently which is mandatory for tetrahedral sorting algorithms like MPVO.

The multi-resolution model is included into a direct volume rendering frame work that adpats the mesh to viewing and classification parameters using a histogram of attribute values and errors.

Future work should make use of a compression scheme for the segments in order to shrink the file sizes of the models. Furthermore, more elaborated segmentation techniques that cluster vertex data based on spatial density and attribute values can be used.

ACKNOWLEDGEMENTS

The authors would like to thank Udo Tremel from EADS Military Aircraft for the beautiful F16-like simulation data, and Uli Bieg from the University of Tübingen for the mesh of the Burdigalian Seaway. This work is supported by EC within FP6 under Grant 511568 with the acronym 3DTV. This work has been funded by the SFB grant 382 of the German Research Council (DFG).

REFERENCES

[CCM+00] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings of the conference on IEEE Visualization '00*, pages 85–92, 2000.

[CDFL+04] P. Cignoni, L. De Floriani, P. Lindstrom, V. Pascucci, J. Rossignac, and C. Silva. Multi–resolution modeling, visualization and streaming of volume meshes. *Eurographics '04 Tutorial Notes*, 2004.

[CGG+04] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3):796–803, 2004.

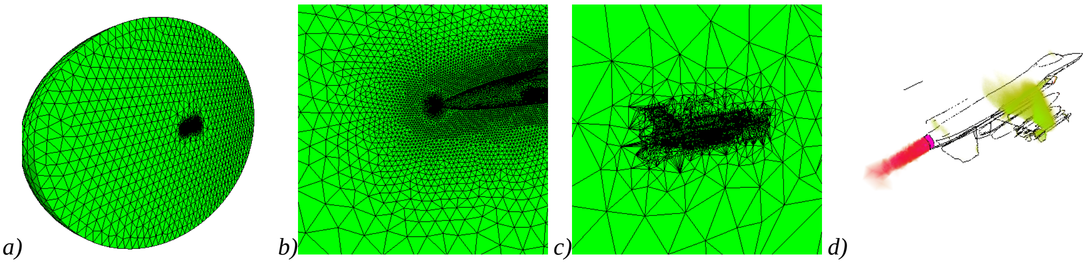


Figure 8: *This model is used for a CFD simulation of a F16-like aircraft and contains about 6 million tetrahedra. Figure (a) shows the full resolution mesh and (b) shows a zoom into the full resolution mesh. Note how the size of the tetrahedra varies which needs to be captured by the segmentation. Figure (c) shows how the mesh is adapted to the viewpoint in the volume rendering (d).*

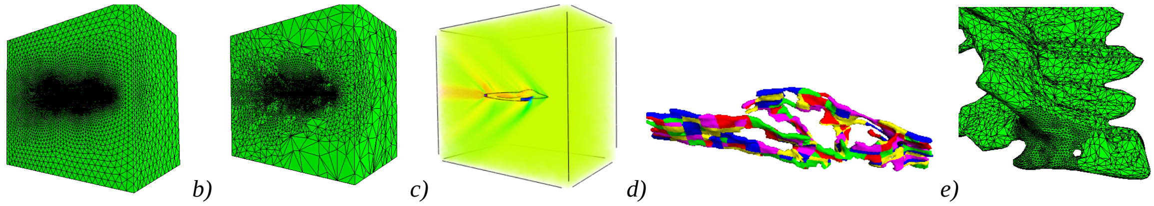


Figure 9: *The NASA fighter dataset (a-c) shows a fighter in a wind tunnel and contains 1.5 million tetrahedra. The full resolution mesh (a) is adpated to the viewpoint and the classification (b) in order to render picture (c) with 250,000 tetrahedra interactively. The Rbl dataset (d) is a portion of an endoplastic reticulum in a cell. Its 3.8 million tetrahedra are simplified to 260,000 tetrahedra using the segmentation in (d). Picture (e) shows a zoom to an adaption of the mesh.*

[CL03] Y. Chiang and X. Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. In *Computer Graphics Forum (Special Issue for Eurographics '03)*, volume 22, pages 493–504, 2003.

[CM02] Prashant Chopra and Jörg Meyer. Tetfusion: an algorithm for rapid tetrahedral mesh simplification. In *Proceedings of the conference on Visualization '02*, pages 133–140, 2002.

[CMRS03] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9:525–537, Nov 2003.

[DDFM+04] E. Danovaro, L. De Floriani, P. Magillo, E. Puppo, D. Sobrero, and N. Sokolovsky. A compact data structure for level–of–detail tetrahedral meshes. Technical report, University of Genova, 2004.

[GI03] Stefan Gumhold and Martin Isenburg. Out-of-core compression for gigantic polygon meshes. In *ACM Transactions on Graphics*, volume 22, pages 935–942, 2003.

[GZ05] Michael Garland and Yuan Zhou. Quadric-based simplification in any dimension. *ACM Transactions on Graphics*, 24(2), 2005.

[Hop96] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[IL05] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *IEEE Visualization 05*, pages 231–238, 2005.

[KE00] M. Kraus and T. Ertl. Simplification of nonconvex tetrahedral meshes. *Electronic Proceedings of NSF/DoE Lake Tahoe Workshop for Scientific Visualization*, 2000.

[KQE04] M. Kraus, W. Qiao, and D. Ebert. Projecting tetrahedra without rendering artifacts. In *Proceedings of IEEE Visualization '04*, pages 27–34, 2004.

[KSE04] T. Klein, S. Stegmaier, and T. Ertl. Hardware–accelerated reconstruction of polygonal isosurface representations on unstructured grids. In *Proceedings of Pacific Graphics '04*, pages 186–195, 2004.

[MHC90] N. L. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, 1990.

[PH97] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH*, pages 217–224, 1997.

[RKE00] S. Roettger, M. Kraus, and T. Ertl. Hardware–accelerated volume and isosurface rendering based on cell projection. In *IEEE Proceedings Visualization '00*, pages 109–116, 2000.

[RO96] Kevin J. Renze and James H. Oliver. Generalized unstructured decimation. *IEEE Computer Graphics and Applications*, Nov 1996.

[SG98] O. G. Staadt and M. H. Gross. Progressive tetrahedralizations. In *Proceedings of IEEE Visualization '98*, pages 397–402, Oct 1998.

[ST90] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Computer Graphics (San Diego Workshop on Volume Visualization)*, 5(4):63–70, 1990.

[VCL+05] Hyu Vo, Steven Callahan, Peter Lindstrom, Valerio Pascucci, and Claudio Silva. Streaming simplification of tetrahedral meshes. Technical report, LLNL technical report UCRL-CONF-208710, 2005.

[Wil92] Peter L. Williams. Visibility-ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.

[YLPM05] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious mesh layouts. In *ACM Transactions on Graphics (SIGGRAPH)*, pages 886–893, 2005.