EXTENDED ABSTRACTS

NSF/DoE Lake Tahoe Workshop

on

Hierarchical Approximation and Geometrical Methods for Scientific Visualization

Granlibakken Conference Center
Tahoe City, California, U.S.A.
October 15–17, 2000

Workshop Co-Chairs:

Hans Hagen, Universität Kaiserslautern, Germany
Gerald Farin, Arizona State University, Tempe, U.S.A.
Bernd Hamann, University of California, Davis, U.S.A.

ACKNOWLEDGEMENTS

The National Science Foundation (NSF) and the Department of Energy (DoE) supported a two-day workshop addressing hierarchical visualization methods. The workshop, held October 15–17, 2000 at the Granlibakken Conference Center, was brought together leading experts developing hierarchical data representation/approximation methods with applications in massive data exploration and visualization. Hierarchical methods emphasizing geometrical approaches were of particular interest. This electronic document contains the (extended) abstracts that the co-organizers solicited from (co-)authors who had also submitted initial one-page abstracts and whose submissions were chosen for presentation.

The workshop participants were scientists, engineers and applied mathematicians, including a large percentage of post-doctoral fellows and Ph.D. students, interested in issues pertaining the representation, storage, and interactive exploration of massive scientific data sets. The workshop addressed fundamental and crucial questions that researchers are facing today. Questions addressed by the workshop included: What are appropriate methods for the efficient construction of compact hierarchical data formats/representations for very large data sets? How can one utilize these formats for interactive data exploration? What are the requirements and solution strategies for collaborative and remote visualization environments? How can hierarchical methods enable more efficient and effective data exploration in virtual and immersive environments? How can geometrical approaches be utilized and extended to help solve some of the pressing problems in massive data compression and simplification?

One of the main goals of the workshop was to involve doctoral students and post-doctoral fellows. The workshop was also a great success in this regard: About half of the talks were presented by students and post-docs. Without the support of the NSF and DoE this exciting workshop would not have been possible. We thank both agencies.

Gerald E. Farin, Arizona State University, Tempe, U.S.A.
Hans Hagen, Universität Kaiserslautern, Germany
Bernd Hamann, University of California, Davis, U.S.A.

LIST OF PRESENTATIONS

Invited Presentations

- **Herbert Edelsbrunner**, Duke University: **Hierarchical Morse Complexes**
- **Andrew J. Hanson**, Indiana University, with Philip Chi-Wing Fu: **Approaches to Interactive Visualization of Large-scale Dynamic Astrophysical Environments**
- **Kenneth I. Joy**, University of California, Davis: **Using Isosurface Methods for Visualizing Solids Defined by Multivariate Functions**
- **Gregory M. Nielson**, Arizona State University: **Multiresolution Models by Adaptive Fitting**

Contributed Talks

- **Ashish Amresh**, Arizona State University, with Gerald Farin and Anshuman Razdan: **Adaptive Subdivision Schemes for Triangular Meshes**
- **Martin Bertram**, Universität Kaiserslautern, Germany, and Lawrence Livermore Nat'l. Lab., with Bernd Hamann, Kenneth I. Joy, Shirley Konkle, and Hans Hagen: **Terrain Modeling Using Voronoi Hierarchies**
- **Peer-Timo Bremer**, University of Hannover, Germany, and University of California, Davis, with Oliver Kreylos, Bernd Hamann, and Franz-Erich Wolter: **Simplification or Large, Closed Triangulated Surfaces Using Atomic Envelopes**
- **Chu-Fei Chang**, State University of New York at Stony Brook, with Amitabh Varshney and Q. Jeffrey Ge: **Hierarchical Image-based and Polygon-based Rendering for Large-scale Visualizations**
- **Paolo Cignoni**, IEI-CNR, Italy, with Leila De Floriani, Paolo Magillo, Enrico Puppo, and Roberto Scopigno: **Volume Visualization of Large Tetrahedral Meshes on Low-cost Platforms**
- **Mark A. Duchaineau**, Lawrence Livermore Nat'l. Lab., with Martin Bertram, Kenneth I. Joy, and Bernd Hamann: **Realtime View-dependent Optimization of Billion-Triangle Isosurfaces**
- **Bjoern Heckel**, Epicentric, Inc.: **Clustering-based Multiresolution Methods for Scientific Visualization**

- **Andreas Hubeli**, ETH Zurich, Switzerland, with Kuno Meyer and Markus Gross: **Mesh Edge Detection — Feature Extraction from Meshes**
- **David N. Kenwright**, Massachusetts Institute of Technology, with Issac J. Trotts and Robert Haimes: **Feature-based Data Reduction in Vector Fields Using Singular Streamlines and Surfaces**
- **Martin Kraus**, University of Stuttgart, Germany, with Thomas Ertl: **Simplification of Nonconvex Tetrahedral Meshes**
- **Oliver Kreylos**, University of California, Davis, and Lawrence Berkeley Nat'l. Lab., with Kwan-Liu Ma and Bernd Hamann: **Point-based Rendering of Arbitrary-mesh Volumetric Data at Multiple Levels of Resolution**
- **Eric C. LaMar**, University of California, Davis, and Lawrence Livermore Nat'l. Lab.: **Efficient Error Calculation for Multiresolution Volume Visualization**
- **Terry J. Ligocki**, Lawrence Berkeley Nat'l. Lab., with Brian Van Straalen, John M. Shalf, Gunther Weber, and Bernd Hamann: **A Framework for Visualizing Hierarchical Computations**
- **Joerg Meyer**, Mississippi State University, with Ragnar Borg, Bernd Hamann, Kenneth I. Joy, and Arthur J. Olson: **VR-based Rendering Techniques for Large-scale Biomedical Data Sets**
- **Valerio Pascucci**, Lawrence Livermore Nat'l. Lab., **Multi-resolution Indexing for Out-of-core Adaptive Traversal of Regular Grids**
- **Philip J. Rhodes**, University of New Hampshire, with R. Daniel Bergeron and Ted M. Sparr: **A Data Model for Multiresolution Scientific Data Environments**
- **Robert Schneider**, Max-Planck-Institute for Computer Science, Saarbrücken, Germany, with Leif Kobbelt and Hans-Peter Seidel: **Geometric Fairing of Irregular Meshes Using Mesh Hierarchies**
- **Han-Wei Shen**, Ohio State University, with Udeepta Dutta Bordoloi: **Hierarchical LIC for Vector Field Visualization**
- **David E. Sigeti**, Los Alamos Nat'l. Lab., with Benjamin F. Gregorski, John J. Ambrosiano, Gerald Graham, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy: **Approximating Material Interfaces During Hierarchical Data Simplification**
- **Georgios Stylianou**, Arizona State University, with Gerald Farin: **Crest Line Extraction from 3D Triangulated Meshes**

- **Xavier Tricoche**, University of Kaiserslautern, Germany, with Gerik Scheuermann, Hans Hagen, and Stephan Clauss: **Scaling the Topology of Symmetric, Second-order Tensor Fields**

- **Issac J. Trotts**, Massachusetts Institute of Technology, with Bernd Hamann, Kenneth I. Joy, and David N. Kenwright: **Simplification of Tetrahedral Meshes Using a Quadratic Error Metric**

- **Gunther Weber**, University of Kaiserslautern, Germany, and Lawrence Berkeley Nat'l. Lab., with Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, and Bernd Hamann: **Crack-free Isosurfaces for AMR Data**

- **David F. Wiley**, University of California, Davis, and Lawrence Livermore Nat'l. Lab., with Martin Bertram, Benjamin W. Jordan and Bernd Hamann: **Hierarchical Best Linear Spline Approximation**

Hierarchical Morse Complexes

Herbert Edelsbrunner
Computer Science, Duke University

In this talk we consider the computational problem of extracting and expressing the structure imposed on a manifold by a Morse function defined over that manifold. Take for example a terrain specified by a smooth height function $h : \mathbb{S}^2 \to \mathbb{R}$ over the 2-sphere (e.g. the surface of the earth). The *critical points* of h are minima, saddles, and maxima, and their stable and unstable manifolds decompose $\mathbb{S}^2$ into the cells of what we call the *Morse complex* of $\mathbb{S}^2$ and h. Apart from constructing that complex, we are interested in defining and constructing a hierarchy, where we have a Morse complex for each scale level. Levels are distinguished by measuring the persistence of critical points and suppressing the ones with persistence less than the level threshold. In a nutshell, the persistence of a maximum is its height advantage over the highest saddle around it, the persistence of a minimum is its depth advantage over the lowest saddle around it, and the persistence of a saddle is that of its associated minimum or maximum. Mathematically, we can imagine climbing up in the hierarchy by annihilating critical points in pairs. Computationally, we may do exactly that by applying a local smoothing operation, or we may choose to simplify by applying the corresponding local transformation to the Morse complex.

The project outlined above requires new ideas on a number of problems, and it is probably best to describe these problems and their solutions in isolation, and to combine the pieces in the end. This talk is about work in progress. For some of the problems we have complete solutions including software, and for others we do not. The pieces we use to form a complete solution are

A Betti numbers for filtrations.

B Persistent homology groups and Betti numbers.

C Simulation of smoothness on a piecewise linear manifold.

D Combinatorial and numerical Morse complexes.

We think we understand Problems A and B completely, also for filtrations of three-dimensional simplicial complexes in $\mathbb{S}^3$. These related to Morse functions $\delta : \mathbb{S}^3 \to \mathbb{R}$, as produced by MRI and other density measuring devices. The numerical challenges in Problems C and D seem significantly more difficult than for height functions over 2-manifolds, but we are optimistic that through tight coupling of combinatorial and numerical computations we can construct structures that are at the same time topologically consistent and numerically most plausible.

Approaches to Interactive Visualization of Large-scale Dynamic Astrophysical Environments

Andrew J. Hanson Philip Chi-Wing Fu

Indiana University, Bloomington, Indiana, USA
Email: {hanson, cwfu}@cs.indiana.edu

Abstract

Dynamic astrophysical data require visualization methods that handle dozens of orders of magnitude in space and time. Continuous navigation across large scale ranges presents problems that challenge conventional methods of direct model representation and graphics rendering. The frequent need to accommodate multiple scales of time evolution, both across multiple spatial scales and within single spatial display scales, compounds the problem because direct time evolution methods may also prove inadequate.

We discuss systematic approaches to building interactive visualization systems that address these issues. The concepts of homogeneous power coordinates, pixel-driven environment-map-to-geometry transitions, and hierarchical antialiased moving-object representations are suggested to handle large scales in space and time. Families of techniques such as these can then support the construction of a virtual dynamic Universe that is scalable, navigable, dynamic, and extensible. Finally, we describe the design and implementation of a working system based on these principles, along with examples of how our methods support the visualization of complex astrophysical effects such as causality and the Hubble expansion.

Keywords: visualization, virtual reality, astrophysics, cosmology

1 Introduction

Exploring large-scale data sets that have a time component requires special attention to spatial and temporal scaling as well as representation issues. We describe a multilevel approach to static data sets that are not necessarily large in the quantity of data, but are large in the number of orders of magnitude of spatial scale required in the representations, as well as addressing the additional challenges of animation and time evolution, visualizing dynamics across many time scales, and integrating the resultant displays with large spatial scale ranges.

1.1 Challenges of Large-scale Dynamic Astrophysics Visualization

To navigate through large-scale dynamic astrophysical data sets, we must overcome difficulties such as the following:

- **Spatial magnitude.** Our data sets sweep through huge orders of magnitude in space. For instance, the size of the Earth is of order 10^7 meter while its orbit around the Sun is of order 10^8 meter. However, the bright stars in the night sky, the stars in the galaxy near our Solar System, go out to about 10^{15} m, while the visible Universe may stretch out to 10^{27} m. Having a unified system to navigate through all these data sets is difficult.

- **Temporal magnitude.** Besides huge orders of magnitude in space, we encounter huge orders of magnitude in time. It takes one day for the Earth to rotate once on its axis, while it takes one year to orbit around the Sun, and Pluto takes almost 248 years. In contrast, our galaxy takes 240 million years to complete one rotation. Thus, handling motion and animation timing properly in astronomical visualization requires special attention.

- **Reference Frames.** There is no absolute reference center for the Universe. We cannot have the same reference frame fixed to the Earth as it moves around the Sun and to the Sun as it moves through the Milky Way. In a dynamic astronomical environment, nothing is at rest. Neither the Ptolemaic Model nor the Copernican model is adequate to simulate the dynamic interaction among data sets because we can place the observer anywhere.

In the next three sections, we discuss techniques for dealing with the above problems: large spatial scale, large time scale, and consistent navigation in dynamic large-scale environments. Then, we present our system design and implementation that addresses these issues.

2 Large Spatial Scales

2.1 Homogeneous Power Coordinate in Space

To handle large spatial scales, we use the homogeneous power coordinates in . Points and vectors in three-dimensional space, say $P = (X, Y, Z)$, are represented by homogeneous power coordinates, $p = (x, y, z, s)$, such that

$$\begin{aligned} s &= \log_k ||P|| \\ (x, y, z) &= (X/k^s, Y/k^s, Z/k^s) \end{aligned}$$

where k is the base. In practice, k is normally 10.

In this way, we can separate the original physical representation into its order of magnitude, the log scale (s), and its unit-length position (or direction) (x, y, z) relative to the original representation. This formulation helps us not only to avoid hitting the machine precision limit during calculation, but also gives us a systematic way to model data sets at different levels-of-detail (LOD) and position them in our virtual environment relative to the current viewing scale.

2.2 Navigating Large-scale in Space

In our system, we define the navigation scale, s_{nav}, as the order of magnitude at which we are navigating through space; that is, one unit in the virtual environment is equivalent to $10^{s_{\text{nav}}}$ m in our real Universe. Consequently, traveling through one unit in the virtual environment gives the sensation of traveling through $10^{s_{\text{nav}}}$ m in the physical Universe. We can thus adapt the step-size to an exploration of the Earth, the Solar System or the whole galaxy. Note that we typically ignore the distortion effects of special relativity, but not the causal effects, during such navigation; this is consistent with rescaling the speed of light to the current virtual scale.

2.3 Levels-of-detail in Space

Besides assisting navigation, we employ s_{nav} to select different levels-of-detail (LOD) in space. The use of LODs to select rendered graphics objects relative to a display pixel has been explored by Hitchner, Astheimer, Maciel, and Reddy. . , under certain conditions when the scale of a data set is greater than s_{nav}, the data set can be represented as an environment map with minimal perception error.

On the other hand, when the scale of a data set is small compared to s_{nav}, we can pick up a smaller scale space-LOD and use that to represent the data set. In addition, when the object size is smaller than one pixel on the screen, we can ignore its rendering unless it is for some reason "flagged" as an important object that warrants an out-of-proportion icon (this is a classic map-making strategy). Details concerning space-LODs and their relation to s_{nav} are given in the *Rotating Scale algorithm* of section 4.

3 Large Time Scales

3.1 Representing Large Scale Ranges and Steps in Time

To handle large scales in time, we developed a log scale technique based on the same general concepts as our spatial scaling. In parallel to the spatial scale s_{nav}, we can define an animation scale, t_{nav}; quantitatively, this time variable specifies the equivalence between one second of observer's screen time (the wall clock) and $10^{t_{\text{nav}}}$ seconds in the simulated Universe. That is, if t_{nav} is 7.4988, we could observe the Earth in the virtual Universe going around the sun once each second. Using this rule, we can determine the observable period of motion, t' (in seconds), for a specified t_{nav} with the formulas:

$$t' = 10^{t_{\text{motion}} - t_{\text{nav}}}$$

Consequently, if we want to visualize the rotation of the Earth on its axis, we can reduce t_{nav} from 7.4988 to 4.9365, which corresponds to a time scale of one day.

3.2 Levels-of-Detail in Time

For a given value of t_{nav}, the Earth will orbit too slowly around the Sun in viewer time, while the moon will orbit too fast. In the former case, we still want to visualize Earth's *direction* of motion even if there is no apparent movement; in the latter case, we still want to see the subset of space (the orbit) occupied by the moon over time, and we still want to see the other planets move in screen time. To have a richer visualization for these "too slow" and "too fast" situations, we can once again make use of t_{motion} and t_{nav}. To handle "too fast" motions, we represent the path as a comet tail. For "too slow" motions, we display a little arrow indicating its direction of motion.

Motion Blur Model. Certain common astronomical models such as planets and moons not only move in their orbits, but spin on their axes. When the object's surface is represented by a complex texture, this rotation creates a texture representation problem exactly analogous to the too-slow/too-fast orbit problem. Whenever the textured object

reaches a time scale where the texture is moving by a large amount in a unit of screen time, the motion loses smoothness and undesirable effects such as stroboscopic aliasing become apparent. We handle this problem by adopting a texture-based motion blur method (see Figure 3) that gives the object a smoother appearance as it speeds up as well as eliminating stroboscopic effects. For very fast rotation speeds, the texture turns into blurred bands exactly analogous to the satellite trails.

4 Modeling our Virtual Universe

Using our scalable representations for space and time variables, we are able to model the full range of physical scales in our virtual Universe. In this section, we first examine the concept of scalable data sets, and then show how we connect scalable models together to form our principal internal data structure, the Interaction Graph. This graph enables us to represent a dynamic virtual Universe that is scalable, navigable, and extensible.

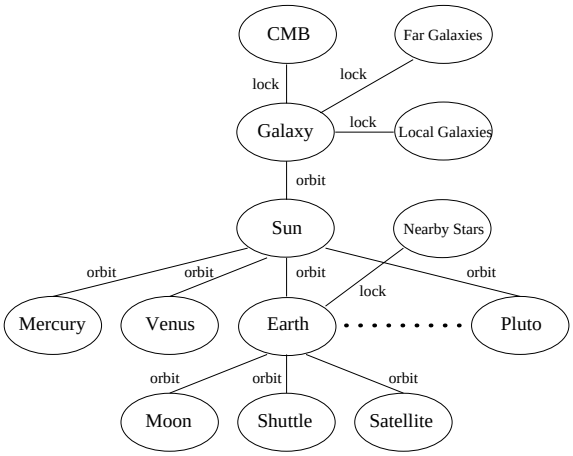


Figure 1. Interaction Graph for the Universe data set collection.

In the interaction graph, each data set is a node and node-node interactions are represented as edges. Figure 1 depicts a typical interaction graph. Note that this interaction graph differs from the scene graph structure (the tree structure traversed in the rendering process). No particular data set is singled out as the root node; the interaction graph serves mainly to 1) organize the data sets by their interactions, 2) facilitate scaling, and 3) allow cascaded animation update.

5 Results

Time-Adjusted Representations. Figure 2 in the color plate section shows a series of Solar System models for different values of t_{nav}. Going from Figure 2(a) to Figure 2(c), we can see that the length of comet tails (Pluto, Neptune and Uranus) shorten as t_{nav} goes from 10.0 down to 9.0; finally, when t_{nav} reaches 6.0, these planets move too slowly to have apparent motions in user time, and the comet tails are replaced by arrows representing long-term motion. Note that all images were captured during real-time system operation.

Motion Blur Representations. In Figure 3 we show a family of representations of a rotating textured object, in this case the Earth. For rotation speeds that are commensurate with screen time, the full detailed texture is appropriate. However, when the animation speeds up, the detailed texture would exhibit undesirable stroboscopic effects; progressively motion-blurring the texture creates an intuitively appealing visualization of the animation.

The Lightcone Clock. Finally, we introduce the *Lightcone Clock*, which symbolizes the visible volume of a single view frustum of spacetime; since the velocity of light is finite, each slice in distance is also a picture taken at one particular instant of time. The correspondence between the time when the currently-observed light was emitted from each disk and the emitter's distance from us in comoving coordinates motivates our describing this visualization as a *clock*.

In order to provide a more effective summary viewpoint of the datasets in the Lightcone Clock, we have adopted a logarithmic scale. The canonical viewing origin on the Earth is at the bottom tip of the Lightcone, and the final upper surface is the event horizon for the first visible radiation, the Cosmic Microwave Background. Figure 4(a) shows the shape of the Lightcone when we use comoving coordinates. When we take the Hubble expansion into account by incorporating the effective radius $a(t)$, the Universe in physical coordinates shrinks with the size of $a(t)$; at the upper (CMB) end of the Hubble-corrected Lightcone Clock, the Universe was less than 1/1000 of its present size. Figure 4(a-e) illustrates the morphing from the comoving-coordinate Lightcone to the $a(t)$-rescaled physical-coordinate Lightcone.

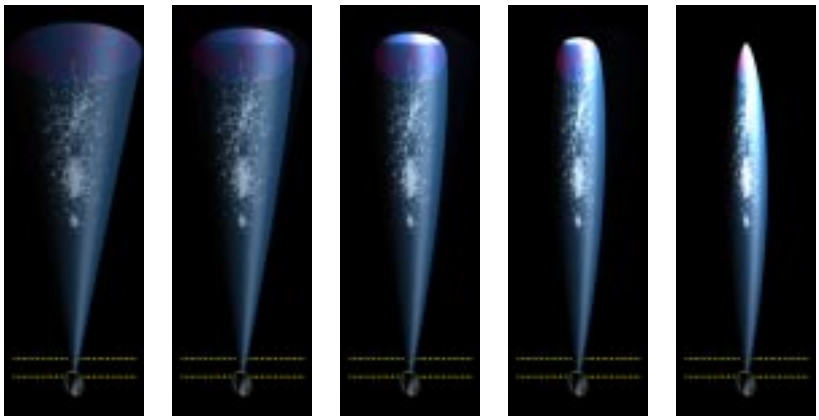
This research was supported by NASA grant number NAG5-8163. We are grateful for the extensive help and participation of Eric A. Wernert and the generosity of the Indiana University Advanced Visualization Laboratory. We thank P.C. Frisch, S. Carroll, D. York, D. Eisenstein, and E. Kolb of the University of Chicago for their assistance with astrophysics and cosmology issues.

(a) $t_{\text{nav}} = 10.0$ (b) $t_{\text{nav}} = 9.0$ (c) $t_{\text{nav}} = 6.0$

Figure 2. "Too fast" representations at (a) roughly 300 years per screen second and (b) roughly 30 years per screen second. (c) The "too slow" representation for the motion of planets in our Solar System at a scale of approximately 10 days per screen second.

(a) $t_{\text{nav}} \leq 4.4$ (b) $4.4 < t_{\text{nav}} \leq 4.9$ (c) $4.9 < t_{\text{nav}}$

Figure 3. Motion Blur Representation : (a) Normal texture. (b) Texture blur for one rotation in two screen seconds. (c) Texture blur for one rotation in less than one screen second.



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

Figure 4. Morphing the Lightcone from Comoving Coordinates to $a(t)$-rescaled Physical Coordinates.

Using Isosurface Methods for Visualizing Solids defined by Multivariate Functions.

Kenneth I. Joy*

Center for Image Processing and Integrated Computing
Department of Computer Science
University of California, Davis

Abstract

We present a method for calculating the envelope surface of a parametric solid object swept along a path in three-dimensional space. The boundary surface of the solid is the combination of parametric surfaces and an implicit surface where the Jacobian of the defining function has a rank-deficiency condition. Using the rank deficiency condition, we determine a set of square sub-Jacobian determinants that must all vanish simultaneously on the implicit surface. When the generator of the swept surface is a trivariate tensor-product B-spline solid and the path is a B-spline curve, we can give a robust algorithm to determine the implicit surface. This algorithm is based upon the "marching tetrahedra" method, which is adapted to work on 4-simplices. The envelope of the swept solid is given by the union of the parametric and implicit surfaces.

Keywords: swept surface; envelopes; boundary surface determination; trivariate B-Spline solids; Jacobian determinant; marching tetrahedra.

1 Introduction

Geometric modeling systems for computer graphics and visualization applications allow the construction of complex models of objects based on simple geometric primitives. As the needs of these systems become more involved, it is necessary to expand the inventory of geometric primitives and design operations and to adapt rendering methods to handle these new primitives. The primary primitive for complex surface definition is the bivariate patch. While complex interfaces have been developed that allow manipulation of these patches, they are not sufficient to describe many surface types. There is a need to investigate alternative representation schemes that use multivariate techniques to define shape.

*joy@cs.ucdavis.edu

In this paper, we examine solid types: the trivariate B-spline solid, and the solid generated by sweeping an object along a curve. We find that the boundary surfaces of these solids, neccesssary to render the solids accurately, can be generated similarly. In short, the boundaries are a combination of parametric surfaces (the images of the boundaries of the domain space) and an implicit surface that is defined where the Jacobian of the defining function has rank $<=$ two.

The parametric surfaces are trivial to render directly, as they are just B-spline patches. It is the implicity surface that is difficult to define, and this is the subject of this paper.

Our algorithms use interval methods to bound the determinants of 3×3 Jacobians. These interval methods are discussed in Section 2. Methods for the trivariate solids are given in Section 3. Extending these methods to swept solids is then straightforward, and are given in Section 4. In each case, the implicity surface is defined as an "isosurface problem".

2 Cone Approximations to Jacobian Determinants

Given a trivariate B-spline function $p(u, v, w)$, the Jacobian of p is defined to be

$$\begin{aligned}\mathcal{J}(\mathbf{p}(u,v,w)) &= \begin{vmatrix} \mathcal{D}_u\mathbf{p}(u,v,w) \\ \mathcal{D}_v\mathbf{p}(u,v,w) \\ \mathcal{D}_w\mathbf{p}(u,v,w) \end{vmatrix} \\ &= \begin{vmatrix} \frac{\partial x}{\partial u}(u,v,w) & \frac{\partial y}{\partial u}(u,v,w) & \frac{\partial z}{\partial u}(u,v,w) \\ \frac{\partial x}{\partial v}(u,v,w) & \frac{\partial y}{\partial v}(u,v,w) & \frac{\partial z}{\partial v}(u,v,w) \\ \frac{\partial x}{\partial w}(u,v,w) & \frac{\partial y}{\partial w}(u,v,w) & \frac{\partial z}{\partial w}(u,v,w) \end{vmatrix} \\ &= \mathcal{D}_u\mathbf{p}(u,v,w) \cdot (\mathcal{D}_v\mathbf{p}(u,v,w) \times \mathcal{D}_w\mathbf{p}(u,v,w))\end{aligned}$$

We will be interested in the values of u, v, and w where $\mathcal{J}(\mathbf{p}(u,v,w)) = 0$. It is straightforward to show that this happens if and only if the triple scalar product

$$\mathcal{D}_u\mathbf{p}(u,v,w) \cdot (\mathcal{D}_v\mathbf{p}(u,v,w) \times \mathcal{D}_w\mathbf{p}(u,v,w)) = 0$$

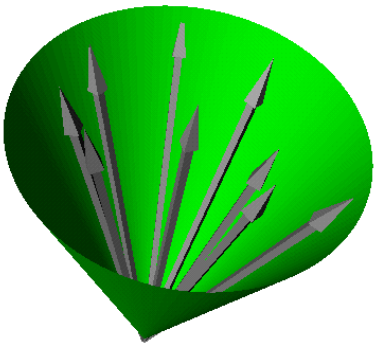


Figure 1. A cone approximation to a set of unit vectors.

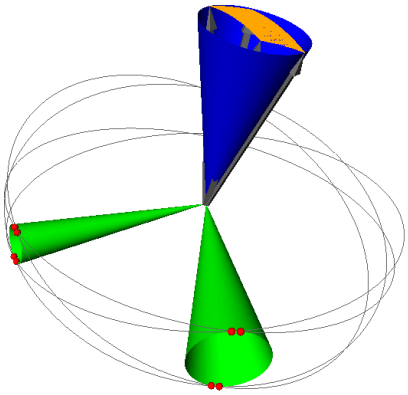


Figure 2. The cross-product cone. The blue cone is the smallest cone that surrounds the cross products of vectors in the two green cones. This cone can be directly calculated by bounding the four vectors that form the normals to the four great circles of the unit sphere tangent to the green cones. The yellow region is the actual region spanned by the cross products calculated from vectors in the two cones.

or, equivalently, when the three vectors $\mathcal{D}_u\mathbf{p}(u,v,w)$, $\mathcal{D}_v\mathbf{p}(u,v,w)$, and $\mathcal{D}_w\mathbf{p}(u,v,w)$ are co-planar (*i.e.*, linearly dependent).

Given a set of unit vectors $\vec{v}_1, \vec{v}_2, ..., \vec{v}_n$, we can bound these vectors by a cone $\mathcal{C}$, defined by an axis $\mathbf{a}$ and a "spread" angle α, such that the angle between each vector $\vec{v}_i$ and the axis $\mathbf{a}$ is less than α . A cone gives an "interval" approximation to a set of unit vectors. Each cone can be associated with a region on the unit sphere – the intersection between the cone, with its apex at the origin, and the unit sphere. The construction of a cone that satisfies these properties was described by Sederberg and Meyers [7], and an example is shown in Figure 1. For a general set of vectors, with varying lengths, we determine a cone bounding the unit vectors, which are determined by dividing each of the vectors by its length.

Given two cones $\mathcal{C}_1$ and $\mathcal{C}_2$, we define the scalar product $\mathcal{C}_1 \cdot \mathcal{C}_2$ to be the interval defining the range of scalar products for pairs of vectors taken from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. We can also define the cross product of two cones to be the smallest cone surrounding all cross products of vectors from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively (see [7], and Figure 2).

The convex hull property holds for trivariate B-spline solids, *i.e.*, the solid is contained in the convex hull of its control points. This implies that for a given cell B in the domain, the cones $\mathcal{C}_u$, $\mathcal{C}_v$, and $\mathcal{C}_w$, constructed from the control points of $\mathcal{D}_u\mathbf{p}$, $\mathcal{D}_v\mathbf{p}$, and $\mathcal{D}_w\mathbf{p}$, respectively, bound the range of directions of the respective partials. This implies that a bound on the Jacobian determinant over B is given by

$$\mathcal{J}(\mathbf{p}) = \mathcal{D}_u\mathbf{p} \cdot (\mathcal{D}_v\mathbf{p} \times \mathcal{D}_w\mathbf{p}) \subseteq L\left(\mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)\right) \quad (1)$$

where L is an interval with positive entries[1], defined to be

$$L = [L_{\text{min}}, L_{\text{max}}]$$

where

$$L_{\text{max}} = \max\left\{|\vec{v}_u||\vec{v}_v||\vec{v}_w| : \vec{v}_u \in \mathcal{C}_u, \vec{v}_v \in \mathcal{C}_v, \text{ and } \vec{v}_w \in \mathcal{C}_w, \right\}, \text{ and}$$
$$L_{\text{min}} = \min\left\{|\vec{v}_u||\vec{v}_v||\vec{v}_w| : \vec{v}_u \in \mathcal{C}_u, \vec{v}_v \in \mathcal{C}_v, \text{ and } \vec{v}_w \in \mathcal{C}_w, \right\}.$$

The quantity $L\left(\mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)\right)$ is an interval product, and produces an interval bounding the range of values of $\mathcal{J}(\mathbf{p})$ over B. If L is interval with positive components, it is clear that if $0 \notin \mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)$ then $\mathcal{J}(\mathbf{p}) \neq 0$ in B. Therefore, given a cell B, and a trivariate B-spline solid $\mathbf{p}$ defined over B, we can state that the implicit boundary surface is not contained in B if

$$0 \notin \mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)$$

where $\mathcal{C}_u$, $\mathcal{C}_v$, and $\mathcal{C}_w$ are the bounding cones for $\mathcal{D}_u\mathbf{p}$, $\mathcal{D}_v\mathbf{p}$, and $\mathcal{D}_w\mathbf{p}$, respectively.

3 The Boundary Surfaces of Trivariate Solids

Let $p(u,v,w)$ be a trivariate solid, where $0 \leq u,v,w \leq 1$. To generate the implicit boundary surface, we adaptively

[1] We assume none of our vectors have zero length.

subdivide the domain space, isolating rectilinear cells in the domain where the isosurface $\mathcal{J}(\mathbf{p}) = 0$ lies. We then use an adaptation of an isosurface algorithm We use a priority queue of domain cells, ordered by decreasing values of the widths of the intervals $\mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)$. The full domain space is initially placed on the queue.

When a cell B is removed from the queue, it is subdivided into eight cells $B_1, ..., B_8$ via planes through the center of the cell and parallel to the uv, uw, and vw plane. For each B_i, the cone approximation $\mathcal{C}_u \cdot (\mathcal{C}_v \times \mathcal{C}_w)$ is calculated, yielding an interval. If zero is contained in this interval, the cell is inserted into the queue. If zero is not contained in the interval, the cell is discarded. By this process, we construct a set of cells, keeping the relevant cells in a priority queue. This process is continued until the widths of the intervals of all cells in the queue are less than a prescribed minimum, or the number of cells in the queue reaches a predetermined number.

If the queue becomes empty, then the implicit boundary surface does not exist over the domain space, and the parametric boundary faces represent the boundary surface of the solid.

4 Swept Objects

The definition of a swept object depends on three factors: the specification of the generator – the object to be swept ; the specification of the trajectory – the sweeping path ; and the specification of the orientation of the generator as it progresses along the trajectory. Thus, given a trivariate B-spline generator $\mathbf{g}(u, v, w)$, a B-spline trajectory curve $\mathbf{c}(t)$, and a 3×3 *coordinate frame transformation* $R(t)$, defined over the same domain as $\mathbf{c}$, the swept object $\mathbf{s}$ is defined to be the set of points where

$$\mathbf{s}(u, v, w, t) = \mathbf{c}(t) + R(t)\mathbf{g}(u, v, w) \qquad (2)$$

for some t in the domain of the curve $\mathbf{c}$ and some (u, v, w) in the domain of $\mathbf{g}$. The swept objects defined by this definition are actually quite general. They allow an arbitrary trivariate solid to be swept along a univariate curve, utilizing an arbitrary coordinate frame transformation to define the orientation/distortion of the solid at a point of the curve. If the coordinate frame transformation is expressible in B-spline form, we can utilize both B-spline subdivision and the convex hull property of splines.

If we assume that the domain space for our solids is a 4-dimensional rectangle defined by $0 \leq u, v, w, t \leq 1$, then a superset of the surfaces that make up the boundary of $\mathbf{s}$ includes the following:

- The boundary surfaces of the solid $\mathbf{s}(u, v, w, t)$, which are the images of the boundaries of the domain space $0 \leq u, v, w, t \leq 1$.
- The surface defined where the rank of the "Jacobian" of $\mathbf{s}$ is less than or equal to two.

In our case, the Jacobian is a 4×3 matrix defined by

$$\begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix}$$

Abdel-Malek and Yeh [1] have shown that the rank-Jacobian condition is equivalent to the vanishing of the determinants of the four possible 3×3 sub-Jacobians of J. These sub-Jacobians, J_{123}, J_{124}, J_{134}, and J_{234} are defined as follows:

$$J_{123} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \end{vmatrix}$$

$$J_{124} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{vmatrix}$$

$$J_{134} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{vmatrix}$$

$$J_{234} = \begin{vmatrix} \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{vmatrix}$$

A point $\mathbf{s}(u, v, w, t)$ is on the surface of the envelope, if

$$J_{123} = J_{124} = J_{134} = J_{234} = 0$$

at (u, v, w, t).

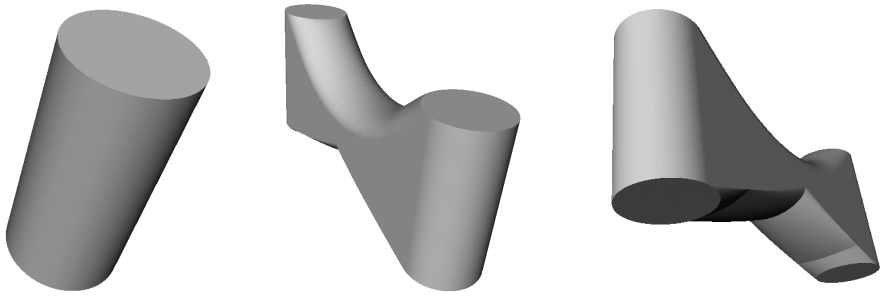
5 The Algorithm

Assuming the $\mathbf{s}(u, v, w, t)$ is given in a B-spline form, we can use a subdivision algorithm to generate the implicit boundary surface of $\mathbf{s}$ where

$$J_{123} = J_{124} = J_{134} = J_{234} = 0.$$

Assuming that the domain space is defined by a four-dimensional rectangle where $0 \leq u, v, w, t \leq 1$, the algorithm proceeds as follows:

- Using subdivision on u, v and w, find a set of cells where $J_{123} = 0$. This can be done by analyzing the trivariate generator $\mathbf{g}$, using the algorithm of Section 3.



(a) (b) (c)

Figure 3. A swept object

- Using subdivision on t, refine the remaining cells, removing those cells where the sub-Jacobians cannot be simultaneously zero.
- Split each four-dimensional cell into 24 4-simplices using CMK-splits, see [5, 6].
- For each 4-simplex, determine a set of tetrahedra that approximates the points where $J_{123} = 0$, see
- For each tetrahedron, determine the isosurface corresponding to $J_{124} = J_{134} = J_{234} = 0$.

The triangles generated through this process define the implicit surface bounding the swept solid $\mathbf{s}$.

We use cone approximations to the partial derivative vectors to get bounds on the individual Jacobian determinants. The strategy is to use subdivision and continually refine the approximations, throwing out those cells where the Jacobian determinants cannot be simultaneously zero. In this way, we obtain a set of small cells in which the envelope surface lies and can use the isosurface methods of the following section to determine the surface.

6 Results

Figure 3 illustrates a cylinder that has been tumbled 180° as it is swept along a linear path. The cylinder has been approximated by a trivariate B-spline, and the coordinate frame vectors have been approximated by B-spline curves.

7 Conclusions

We present an algorithm that generates the boundary surfaces of both trivariate solids and a swept trivariate solids. This algorithm generates the boundary through a combination of parametric and implicit surfaces. Generation of the implicit surface is accomplished through an isosurface routine. We have found that this algorithm can be used to generate a variety of solids. The algorithm depends on the B-spline representation only to calculate Jacobian bounds on each cell and to use subdivision to isolate the cells.

This method generates a superset of the actual boundary of the swept solid, generating some surfaces on the interior of the solid. In future work, we will identify these interior points and eliminate them from the description of the boundary.

References

[1] K. Abdel-Malek and H. Yeh. Geometric representation of the swept volume using jacobian rank-deficiency conditions. *Computer-aided Design*, 29(6):457–468, 1997.

[2] J. Conkey and K. I. Joy. Using isosurface methods for visualizing the envelope of a swept trivariate solid. In *Proceedings of Pacific Graphics 2000*, pages 272–279, Oct. 2000.

[3] K. I. Joy and M. A. Duchaineau. Boundary determination for trivariate solids. In *Proceedings of Pacific Graphics '99*, pages 82–91, Oct. 1999.

[4] C. Madrigal and K. I. Joy. Generating the envelope of a swept trivariate solid. In *Proceedings of the IASTED International Conference on Computer Graphics and Imaging*, pages 5–9, Oct. 1999.

[5] D. Moore. Subdividing simplices. In D. Kirk, editor, *Graphics Gems III*, pages 244–249. Academic Press, 1987.

[6] G. M. Nielson. Tools for triangulations and tetrahedrizations and constructing functions defined over them. In G. M. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 429–525. IEEE Computer Society Press, 1997.

[7] T. Sederberg and R. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5(2):161–171, 1988.

Multiresolution Models by Adaptive Fitting

Gregory M. Nielson

Abstract

The value of multiresolution models in Scientific Visualization and CAGD is well established. This talk will survey the use of adaptive fitting techniques for computing these types of models. The scope of the talk will include the three application areas of height field modeling, volume modeling and surface modeling. We will discuss techniques and show examples of top-down and bottom-up approaches to all three application areas.

Adaptive Subdivision Schemes for Triangular Meshes

Ashish Amresh Gerald Farin
Anshuman Razdan
Arizona State University, Tempe AZ 85287-5106
Email: amresh@asu.edu Phone: (480) 965 7830 Fax: (480) 965 2751.

October 5, 2000

Abstract

Of late we have seen an increase in the use of subdivision techniques for both modeling and animation. They have given rise to a class of surfaces called subdivision surfaces. These have many advantages over traditional Non Uniform Rational B-spline (NURB) surfaces. Subdivision surfaces easily address the issues related to multiresolution, refinement, scalability and representation of meshes. Many schemes have been introduced that take a coarse mesh and refine it using subdivision. They can be mainly classified as Approximating - in which the original coarse mesh is not preserved, or Interpolating - wherein the subdivision forces the refined mesh to pass through the original points of the coarse mesh. The schemes used for triangular meshes are mainly the Loop scheme, which is approximating in nature and the Modified Butterfly scheme which is interpolating. Subdivision schemes are cost intensive at higher levels of subdivision. In this paper we introduce two methods of adaptive subdivision for triangular meshes that make use of the Loop scheme or the Modified Butterfly scheme to get approximating or interpolating results respectively. The results are obtained at a lower cost when compared with those obtained by regular subdivision schemes. The first method uses the dihedral angle to develop an adaptive method of subdivision. The other method relies on user input, i.e., the user specifies which parts of the mesh should be subdivided. This process can be automated by segmentation techniques, for example watershed segmentation, to get the areas in the mesh that need to be subdivided. We compare our methods for various triangular meshes and present our results.

1 Introduction

When Catmull and Clark[1] and Doo and Sabin[2] published their papers little did they expect that subdivision would be used so extensively as it is being used today for the purposes of modeling and animation. It has been used to a large extent in movie production, commercial modelers such as MAYA 3.0, LIGHTWAVE 6.0 and game development engines.

The basic idea behind subdivision can be traced as far back as to the late 40s and early 50s when G. de Rham used *corner cutting* to describe smooth curves. In recent times the application of subdivision surfaces has grown in the field of

computer graphics and computer aided geometric design(CAGD) mainly because it easily addresses the issues raised by multiresolution techniques to address the challenges raised for modeling complex geometry. The subdivision schemes introduced by Catmull and Clark[1] and Doo and Sabin[2] set the tone for other schemes to follow and schemes like Loop[6], Butterfly[3] and Modified Butterfly[14], Kobbelt [4] have become popular. These schemes are chiefly classified as either approximating, where the original vertices are not retained at newer levels of subdivision, or interpolating, where subdivision makes sure that the original vertices are carried over to the next level of subdivision. The Doo-Sabin, Cattmull-Clark and Loop schemes are approximating and Butterfly, Modified Butterfly and Kobbelt schemes are interpolating.
It is seen that all the schemes provide a process of global refinement at every level of subdivision. This can lead to a heavy computational load at higher levels of subdivision. For example, it is observed that in the Loop scheme every level of subdivision increases the triangular count by four. It is also observed that for most surfaces there are regions that become reasonably smooth after few levels of subdivision and only certain areas of the surface where there is a high curvature change need high subdivision levels to make it smooth. It therefore is not ideal to have a global subdivision scheme being applied at every level. Adaptive Subdivision aims at providing a local subdivision rule that governs weather or not a given face in a mesh needs to be subdivided at the next level of subdivision.

2 Existing methods

Mueller[8] proposed an adaptive process for Catmull-Clark and Doo-Sabin subdivision schemes. In his method the adaptive refinement is controlled by the vertices at every level of subdivision. The approximation is carried by an error calculated at every vertex of the original mesh before it is subdivided. This error is the distance between the vertices of the original mesh and their limit point. All the vertices that lie in the error range are labeled differently and special rules are applied for subdividing a polygon when it contains one or more of these labeled vertices.
Xu and Kondo[12] devised an adaptive subdivision scheme based on the Doo-Sabin scheme. In their method the adaptive refinement is controlled by the faces of the original mesh. Faces are labeled as *alive* or *dead* if they have to be subdivided or not. The labeling is based on the dihedral angle i.e. the angle between the normal vectors of adjacent faces and a tolerance limit for this angle is set. If a face satisfies the set tolerance then it is labeled as dead and further refinements are stopped for that face.
Kobbelt has developed adaptive refinement for both his Kobbelt scheme and newly introduced $\sqrt{3}$ subdivision[5]. His refinement strategy ia also centered around the faces. In both his schemes adaptive refinement presents a face cracking problem, discussed later in the paper, that he solves by using a combination of mesh balancing and the Y-technique for his Kobbelt scheme and for his $\sqrt{3}$ subdivision he uses a combination of dyadic refinement, mesh balancing and gap fixing by temporary triangle fans. This process is well known in the finite element community under the

name red-green triangulation [11].
Zorin et. al. have developed adaptive refinement strategies in[15], where they have additional constraints that require a certain number of vertices in the neighborhood of those vertices calculated by adaptive subdivision to be present. Their methods have been implemented on the Loop scheme.
We observe that the adaptive strategies can be developed in two ways, one, by classifying which vertices need to be subdivided (vertex spilt operation at the next level), or two, by identifying those faces that should be subdivided (face split at the next level).

3 The Loop Scheme

The Loop scheme is a simple approximating face-split scheme for triangular meshes proposed by Charles Loop [6].The scheme is based on the triangular splines[10], which produces C^2 - continuous surfaces over regular meshes. A regular mesh is a mesh which has no *extraordinary* vertices. A vertex is not extraordinary in a triangular mesh if it has six adjacent vertices, it also means that the vertex has a valance of six. The Loop scheme produces surfaces that are C^2 - continuous everywhere except at extraordinary vertices, as explained earlier these are the vertices that do not have a valance of six, where they are C^1 - continuous. A boundary vertex is regular or even if it has a valance of three and is extraordinary for any other valance. The masks for the Loop scheme are shown in Figure 1. For boundaries special rules are used. These rules produce a cubic spline curve along the boundary. The curve only depends on control points on the boundary. The scheme works as follows

- for every original vertex a new vertex(odd vertex) is calculated by calculating β from equation (1), where k is the number of adjacent vertices for the vertex, and finding the suitable coefficients for the adjacent control points as shown in Figure 4.
- for every edge in the original mesh a new vertex(even vertex) is calculated by using the mask shown in Figure 1.
- every triangle in the original mesh gives rise to six new vertices, three from original vertices and three from original edges, these six vertices are joined to give four new triangles.

In Figure 1, k is the no of adjacent vertices for a given vertex and β can be chosen as

$$\beta = 1/k \left(5/8 - (3/8 + 1/4cos2\pi/k)^2\right) \tag{1}$$

The value for β [6] was so found that the resulting surface is C^1 - continuous at the extraordinary points. For regular vertices the coefficients for calculating the new vertices are obtained by substituting k as six in the mask for even vertices shown in Figure 1.

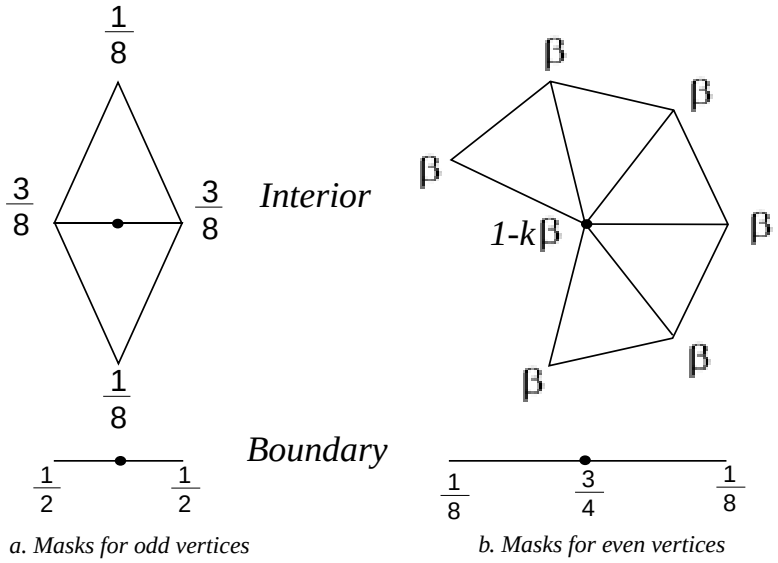


Figure 1: Masks for Loop scheme

4 Our methods

We discuss the methods we have developed for the purposes of adaptively subdividing meshes generated by the Loop scheme. Our first method is based on identifying which faces are *flat* and proposing suitable mesh refinements based on the properties of the of its neighboring faces. The second method is based on user interaction, where the user can select areas on the mesh where refinements are warranted.

4.1 Dihedral angle method

Our first method is called the dihedral angle method because it uses the angles between normals of a face with adjoining face normals to determine if the face needs to be subdivided or not. If the angles are within some tolerence limit then we classify the face as flat. Adaptive process introduces cracking between faces and triangle fans are introduced to solve this problem. We take care of the cracking problem in our scheme by a process of refinement that takes into account the nature of the adjacent faces before refining a given face. The process is shown in Figure 2. We introduce an adaptive weight a that controls the tolerence limit for the angles between the normals. The nature our scheme works is given as follows:

- Normal for each face is calculated
- For every face the angle between its normal and normals of adjoining faces are calculated
- If all angles lie below a certain threshold then the face is set to be flat

- For every face a degree of flatness, which is the number of adjoining faces that are flat, is set. The maximum value for this degree can be three and minimum will be zero. Based on the degree of flatness, refinement is done as shown in Figure 2.

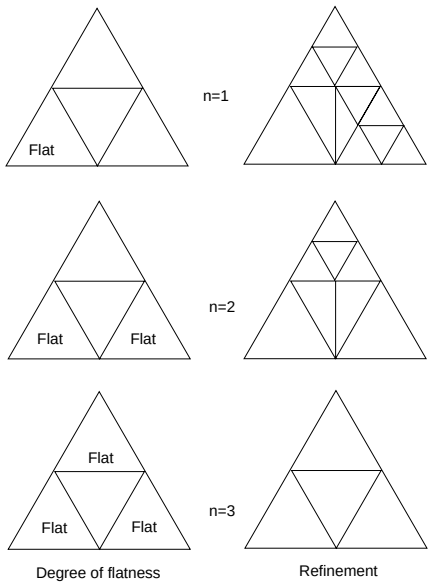


Figure 2: Mesh refinement based on the degree of flatness

4.2 Analysis of results

It is seen that many levels of adaptive refinement produce degenerate triangulation and therefore, we suggest that after a level of adaptive subdivision a suitable realignment of triangles should be done. Realignment introduces some more computation and if speed is important a good strategy could be to alternate between adaptive and normal subdivision methods. We now take a of a cat and compare our adaptive scheme with the normal approximating scheme at two levels of subdivision. Figure 3 shows the comparison of the normal Loop scheme at two levels(left) and using our adaptive scheme and then applying the Loop scheme at two levels of subdivision(right).

4.3 Automated segmentation method

The main drawback with dihedral angle method is that it acts on the whole mesh and goes through a lot of computation to identify the triangle types and find their degree of flatness. If there could be a way that the user identifies the regions in a mesh that needs to be subdivided then these computations can be avoided. This method therefore gives control to the user and is very simple in nature. It asks the

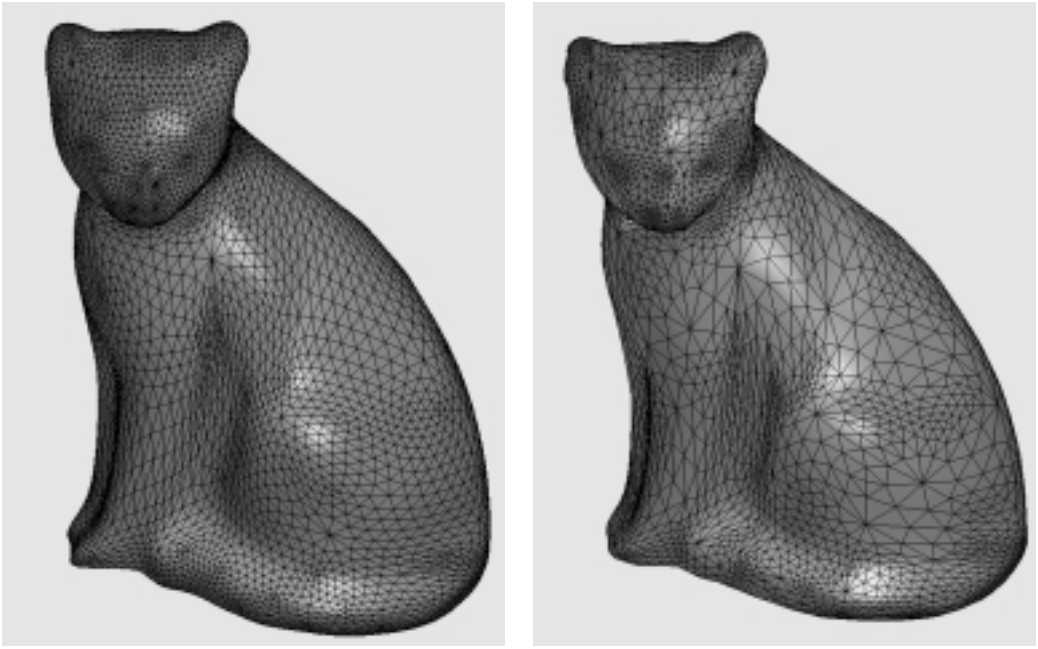


Figure 3: Using dihedral angle adaptive Loop subdivision method on a cat data file, image on the left is normal subdivision and the one on the right is subdivided adaptively

user to define areas on the mesh that need more refinement and the Loop scheme is applied to obtain an adaptive mesh at the next level. The user would also benefit if the method of picking these areas were done by an automated process. Segmentation of meshes deals with segmenting meshes based on their topology and we find that watershed segmentation [7] to be an effective technique to employ before adaptive subdivision. A brief explanation of the watershed segmentation algorithm is as follows:

- Estimate a height function at each vertex of the mesh, the height function used is the curvature calculated at the vertex.
- Based on the height function, the vertices whose curvature is less than the curvatures of its neighbors is labeled as a minima.
- From every vertex that is not a minima, a token is sent in the direction of its neighbor with lowest curvature. The token stops when it hits a minima. The minima is copied into every vertex in the tokens path, and when the token stops every minima establishes its own region.
- Merging of regions is performed if they satisfy certain conditions of similarity as described in [7].

The adaptive scheme now works as follows:

- Apply normal Loop scheme to a coarse mesh till a reasonable resolution is reached. Usually two levels of subdivision are enough.
- Identify regions using the watershed segmentation method.
- Find points along the boundary between segmented regions, and mark all triangles that contain these points.
- Decide on a triangle threshold limit, i.e. the number of adjacent triangles to the triangles on the boundary and mark these triangles.
- Subdivide only the marked triangles.

4.4 Analysis of results

Figure 4 shows the results of adaptively subdividing a simple uniform mesh of a vase data. The areas of refinement happen to be the areas where the curvature changes and these have been identified by the user and for purposes of simplicity we pick some points as depicted in the mesh on the left in Figure 4. The results for a complicated golf head file are shown in Figure 5. In this file the user defined points have been picked by watershed segmentation and this can be viewed as an automated process. The figure also shows a comparison of subdividing the golf head normally by Loop subdivision(left), subdivide it by dihedral angle subdivision(middle) and watershed method(right). It can be seen that for data that has a lot of curvature changes and irregularity the dihedral angle method is a better way of adaptive subdivision as the whole mesh is filled with irregularities and a process like watershed segmentation would have to compute all the various regions. Suitable applications would be in terrain modeling where we find a lot of irregularity in the data. In cases where the regions can be marked with ease and where change in curvature is consistent a segmentation approach should be applied to get faster results as the computational value of the dihedral angle method could be a burden. Suitable applications could be meshes used in character animation and industrial design prototypes.

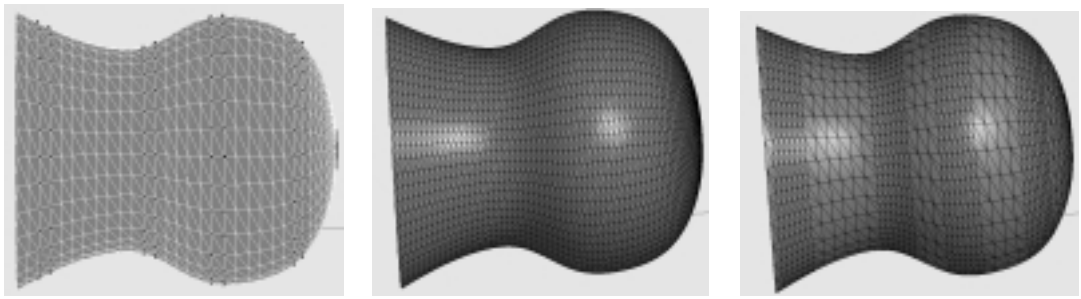


Figure 4: Using automated segmentation method for a vase data file, image on the left is the basemesh with points picked, one in the middle uses Loop subdivision and the one on the right is subdivided adaptively

Figure 5: Comparison of normal subdivision, dihedral angle adaptive subdivision and watershed adaptive subdivision applied on to a golf head file

5 Computational analysis

We compare the cost considerations of the two schemes in this section. As discussed earlier at every level of the Loop scheme the number of triangles generated increases by a factor of four. For the results obtained by dihedral scheme, shown in Figure 3, the base mesh of cat consists of 698 triangles and a second level Loop subdivision produces 9168 triangles as opposed to 3337 triangles produced by dihedral subdivision.
For the results obtained by automated segmentation method, shown in figure 5, the base mesh of the golf head consisted of 33 triangles and a second level Loop subdivision produces 528 triangles, upon which watershed segmentation is used to identify the regions. Continuing with Loop subdivision produces 8448 triangles at the fourth level as opposed to 7173 triangles produced by the automated segmentation method. It can be seen that that saving in the size of the mesh is considerably high while using the dihedral method but it also must be noted that the dihedral method cannot be applied in succession. We also note that when the mesh is undulating in nature i.e. not very consistent in curvature then using a curvature based user defined process like watershed segmentation will not result in considerable savings in mesh size.

6 Future work and conclusions

In this paper we have presented adaptive schemes for triangular meshes and our results have been based on the Loop scheme. These could be very well applied to the Modified Butterfly scheme. These schemes could also be very well extended to subdivision schemes that work on polygonal meshes like Catmull-Clark and Doo-Sabin. Our methods are a shift from the methods proposed in [5],[15] where the additional constraints due to smoothing require extra storage for temporary triangles. Our first method tries to alternate normal and adaptive subdivision to bring in smoothing properties to the resulting surface. For a reasonable angle tolerance the results obtained are in accordance to the results obtained by normal Loop subdivision.

Our second method tries to bring user interaction before an adaptive subdivision scheme is applied. It is seen that automating the interaction by a process like watershed segmentation brings in good results. However watershed segmentation requires meshes with a reasonable amount of resolution, which is a good thing as we need adaptive subdivision for only higher subdivision levels. Subdivision at lower levels can be achieved with good speed by normal subdivision. We therefore propose to have a coarse mesh subdivided by normal Loop scheme for two levels and achieving a reasonable resolution. The watershed algorithm is now run to get the points on the boundary of various segments. The faces lying along the boundary are the only ones subdivided at the next levels of subdivision. The area of applying segmentation along with subdivision has a lot of promise and holds potential for future research.

References

[1] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[2] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, 1978.

[3] N. Dyn, J. Gregory, and D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9:160–169, 1990.

[4] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Proceedings of Eurographics.*, 409-420, 1996.

[5] L. Kobbelt. $\sqrt{3}$ Subdivision. *Computer Graphics Proceedings.*, SIGGRAPH 2000.

[6] C. Loop. Smooth subdivision surfaces based on triangles. *Masters Thesis.*, University of Utah, Dept. of Mathematics, 1987.

[7] A. Mangan and R. Whitaker. Partitioning 3D meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 308-321, oct-dec, 1999.

[8] H. Mueller and R. Jaeschke. Adaptive subdivision curves and surfaces *Proceedings of Computer Graphics International '98*, 48-58, 1998.

[9] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Gr.16(4).*, 1997.

[10] H. Seidel. Polar forms and triangular B-Splines. *Tutorial notes in Pacific Graphics*,61-112, 1997.

[11] M. Vasilescu and D. Terzopoulos. Adaptive meshes and shells: Irregular triangulation, discontinuities and hierarchical subdivision. *Proceedings of the Computer Vision and Pattern Recognition conference*,829-832, 1992.

[12] Z. Xu and K. Kondo. Adaptive refinements in subdivision surfaces. *Eurographics '99, Short papers and demos*, 239-242, 1999.

[13] D. Zorin. Subdivision and multiresolution surface representations. *PhD Thesis.*, Caltech, Pasadena, 1997.

[14] D. Zorin, P. Schroder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. *SIGGRAPH '96 Proceedings*, pages 189-192, 1996.

[15] D. Zorin, P. Schroder, and W. Sweldens. Interactive multiresolution mesh editing. *SIGGRAPH '97 Proceedings*, pages 259-268, 1997.

Terrain Modeling using Voronoi Hierarchies

Martin Bertram[1,2] Bernd Hamann[2] Ken Joy[2] Shirley Konkle[2] Hans Hagen[1]

Abstract. We present a new algorithm for terrain modeling based on Voronoi diagrams and Sibson's interpolant. Starting with a set of scattered data sites in the plane with associated function values defining a height field, our algorithm constructs a top-down hierarchy of smooth approximations. We use the convex hull of given sites as domain for our hierarchical representation. Sibson's interpolant is used to approximate the underlying height field based on associated function values of selected subsets of the data sites. Therefore, our algorithm constructs a hierarchy of Voronoi diagrams for nested subsets of the given sites. The quality of approximations obtained with our method compares favorably to results obtained from other multiresolution algorithms like wavelet transforms. Our Approximations for every level of resolution are C^1-continuous, except at the selected sites, where only C^0-continuity is satisfied. The expected time complexity of our algorithm is $O(n \log n)$ for n sites when applying simple acceleration methods. In addition to a hierarchy of smooth approximations, our method provides a cluster hierarchy based on convex cells and an importance ranking for sites.

1 Introduction

Clustering techniques [7] generate a data-dependent partitioning of space representing inherent topological and geometric structures of scattered data. Adaptive clustering methods recursively refine this partitioning resulting in a multiresolution representation that is required for applications like progressive transmission, compression, view-dependent rendering, and topology reconstruction. For example, topological structures of two-manifold surfaces can be reconstructed from scattered points in three-dimensional space using adaptive clustering methods [6]. In contrast to mesh-simplification algorithms, adaptive clustering methods do not require a grid structure connecting the given data points. A cluster hierarchy is built in a "top-down" approach, so that coarse levels of resolution require less computation times than finer levels.

In this paper, we present a Voronoi-based adaptive clustering method for terrain modeling. Arbitrary samples taken from large-scale terrain models are recursively selected according to their relevance. Continuous approximations of the terrain model are constructed based on the individual sets of selected sites using Sibson's interpolant [10]. We have implemented this algorithm using a Delaunay triangulation, i.e., the dual of a Voronoi diagram, as underlying data structure. Constructing a Delaunay triangulation requires less implementation than constructing the corresponding Voronoi diagram, since a lot of special cases (these where Voronoi vertices have a valence greater than three) can be ignored. A major drawback of Delaunay triangulations is that they are not unique, in general. This becomes evident when the selected sites are sampled from regular, rectilinear grids such that the diagonal for every quadrilateral can be flipped, resulting in random choices affecting the approximation. The corresponding Voronoi diagram, however, is uniquely defined and can instantly be derived from a Delaunay triangulation. Sibson's interpolant is also efficiently computed from a Delaunay triangulation. The advantage of our method compared to Delaunay-based multiresolution methods [5] is that our approximations are unique and C^1-continuous almost everywhere.

[1]University of Kaiserslautern, Germany
[2]University of California at Davis
bertram@informatik.uni-kl.de

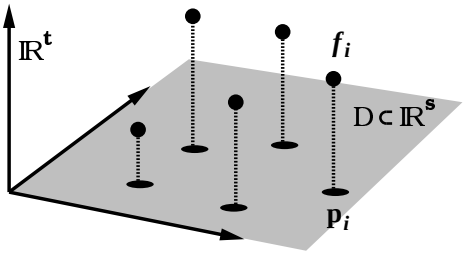


Figure 1: Scattered points with associated function values.

2 Adaptive Clustering Approach

Adaptive clustering schemes construct a hierarchy of regions, each of which is associated with a simplified representation for the data points located inside. We assume that a data set is represented at its finest level of resolution by a set P of n points in the plane with associated function values, $t = 0, 1, \ldots$:

$$P = \{(\mathbf{p}_i,\ f_i) \mid \mathbf{p}_i \in \mathbb{R}^2,\ f_i \in \mathbb{R},\ i = 1, \ldots, n\}.$$

This set P is sampled from a continuous function $f : D \to \mathbb{R}$, where $D \subset \mathbb{R}^2$ is a compact domain containing all points $\mathbf{p}_i$. The points $\mathbf{p}_i$ define the associated parameter values for the samples f_i. We do not assume any kind of "connectivity" or grid structure for the points $\mathbf{p}_i$. For other applications than terrain modeling, the points p_i can have s dimensions with t-dimensional function values f_i, see Figure 1.

The output of an adaptive clustering scheme consists of a number of levels $L_j,\ j = 0, 1, \ldots$, defined as

$$L_j = \{(\tau_k^j,\ \tilde{f}_k^j,\ \varepsilon_k^j) \mid k = 1, \ldots, n_j\},$$

where for every level with index j, the tiles (regions) $\{\tau_k^j \subseteq D \mid k = 1, \ldots, n_j\}$ form a partitioning of the domain D, the functions $\tilde{f}_k^j : \tau_k^j \to \mathbb{R}$ approximate the function values of points located in the tiles τ_k^j, *i.e.*,

$$\tilde{f}_k^j(\mathbf{p}_i) \approx f_i \quad \forall \mathbf{p}_i \in \tau_k^j,$$

and the residuals $\varepsilon_k^j \in \mathbb{R} \geq 0$ estimate the approximation error. In principle, any error norm can be chosen to compute the residuals ε_k^j. We note that the error norm has a high impact on the efficiency and quality of the clustering algorithm, since it defines an optimization criterion for the approximations at every level of resolution. We suggest to use the following norm:

$$\varepsilon_k^j = \left(\frac{1}{n_k^j} \sum_{\mathbf{p}_i \in \tau_k^j} |\tilde{f}_k^j(\mathbf{p}_i) - f_i|^p \right)^{\frac{1}{p}}, \quad p \in [1, \infty], \qquad (1)$$

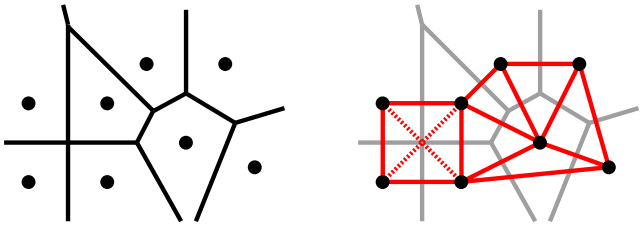


Figure 2: Planar Voronoi diagram and its dual, the (not uniquely defined) Delaunay triangulation.

where $n_k^j = |\{\mathbf{p}_i \in \tau_k^j\}|$ is the number of points located in tile τ_k^j. In the case of $p = \infty$ the residual is simply the maximal error within the corresponding tile.

A global error ε^j with respect to this norm can efficiently be computed for every level of resolution from the residuals ε_i^j as

$$\begin{aligned} \varepsilon^j &= \left(\frac{1}{n} \sum_{k=1}^{n_j} \sum_{\mathbf{p}_i \in \tau_k^j} |\tilde{f}_k^j(\mathbf{p}_i) - f_i|^p \right)^{\frac{1}{p}} \\ &= \left(\frac{1}{n} \sum_{k=1}^{n_j} n_k^j \left(\varepsilon_k^j\right)^p \right)^{\frac{1}{p}}. \end{aligned} \tag{2}$$

Starting with a coarse approximation L_0, an adaptive clustering algorithm computes finer levels L_{j+1} from L_j until a prescribed number of clusters or a prescribed error bound is satisfied. To keep the clustering algorithm simple and efficient, the approximation L_{j+1} should differ from L_j only in cluster regions with large residuals in L_j. As the clustering is refined, it should eventually converge to a space partitioning, where every tile contains exactly one data point or where the number of points in every tile is sufficiently low providing zero residuals.

3 Constructing Voronoi Hierarchies

In the following, we describe our adaptive clustering approach for multiresolution representation of scattered data: a hierarchy of Voronoi diagrams [2, 9] constructed from nested subsets of the original set of points.

The *Voronoi diagram* of a set of points $\mathbf{p}_i,\ i = 1, \ldots, n$ in the plane is a space partitioning consisting of n tiles τ_i. Every tile τ_i is defined as a subset of $\mathbb{R}^2$ containing all points that are closer to $\mathbf{p}_i$ than to any $\mathbf{p}_j,\ j \neq i$, with respect to the Euclidean norm.

A Voronoi diagram can be derived from its dual, the *Delaunay triangulation* [1, 3, 4, 5], see Figure 2. The circumscribed circle of every triangle in a Delaunay triangulation does not contain any other data points. If more than three points are located on a such a circle, then the Delaunay triangulation is not unique. The Voronoi vertices are located at the centers of circumscribed circles of Delaunay triangles, which can be exploited for constructing a Voronoi diagram. The Voronoi diagram is unique, in contrast to the Delaunay triangulation.

A Delaunay triangulation is constructed in expected linear time, provided the points are evenly distributed [8]. Figure 3 illustrates the adaptive construction process in the plane. For every point inserted into a Delaunay triangulation, all triangles whose circumscribed circles contain the new point are erased. The points belonging to erased triangles are then connected to the new point, defining new triangles that automatically satisfy the Delaunay property. Point insertion is an operation performed in expected constant

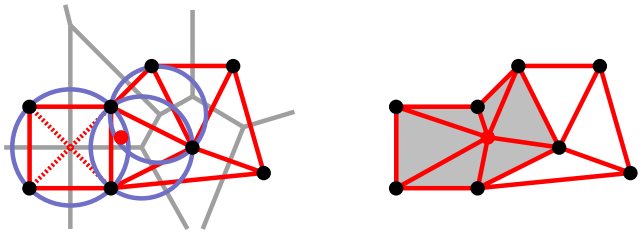


Figure 3: Construction of Delaunay triangulation by point insertion. Every triangle whose circumscribed circle contains the inserted point is erased. The points belonging to removed triangles are connected to the new point.

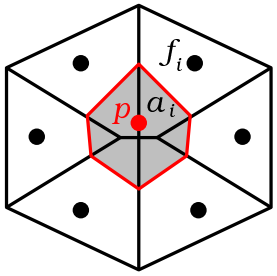


Figure 4: Computing Sibson's interpolant at point p by inserting p into a Voronoi diagram and using the areas cut away from every tile as blending weights.

time, provided that the triangles to be removed are identified in expected constant time, which requires the use of some acceleration method. For applications in k-dimensional spaces ($k > 2$) the Delaunay triangulation consists of k-simplices whose circumscribed k-dimensional hyperspheres contain no other point.

The adaptive clustering algorithm uses *Sibson's interpolant* [10] constructing the functions $\tilde{f}_k^j$. Sibson's interpolant is based on blending function values f_i associated with the points $\mathbf{p}_i$ that define the Voronoi diagram. The blending weights for Sibson's interpolant at a point $\mathbf{p} \in \mathbb{R}^2$ are computed by inserting $\mathbf{p}$ temporarily into the Voronoi diagram and by computing the areas a_i that are "cut away" from Voronoi tiles τ_i, see Figure 4. The value of Sibson's interpolant at $\mathbf{p}$ is defined as

$$f(\mathbf{p}) = \frac{\sum_i a_i f_i}{\sum_i a_i}.$$

Sibson's interpolant is C^1-continuous everywhere except at the points $\mathbf{p}_i$. To avoid infinite areas a_i, the Voronoi diagram is clipped against the boundary of the compact domain D. A natural choice for the domain D is the convex hull of the points $\mathbf{p}_i$.

In the following, we provide the clustering algorithm in pseudocode. The algorithm performs these steps:

(i) Construct the Voronoi diagram for the minimal point set defining the convex hull of all points $\mathbf{p}_i,\ i = 1, \ldots, n$. The tiles of this Voronoi diagram define the cluster regions τ_k^0 of level L_0.

(ii) From the functions $\tilde{f}_k^j$, defined by Sibson's interpolant and from error norm (1) ($p = 2$), compute all residuals ε_k^0. To avoid square root computations, $(\varepsilon_k^0)^2$ is stored.

(iii) Refinement: $L_j \rightarrow L_{j+1}$. Let m be the index of a maximal residual in L_j, *i.e.*, $\varepsilon_m^j \geq \varepsilon_k^j\ \forall k = 1, \ldots, n_j$. Among all $\mathbf{p}_i \in \tau_m^j$, identify a data point $\mathbf{p}_{max}$ with maximal error

No. Voronoi Tiles	Error $(p = \infty)$ [%]	Error $(p = 2)$ [%]
100	31.6	3.13
200	17.1	1.96
300	16.3	1.55
400	13.9	1.33
500	11.9	1.21
1000	10.8	0.80

Table 1: Approximation errors in percent of amplitude for Crater-Lake terrain data set. Figure 5 shows the different levels of resolution.

$max_{\mathbf{p}_i \in \tau_m^j}\{|\tilde{f}_m^j(\mathbf{p}_i) - f_i|\}$. Insert $\mathbf{p}_{max}$ into the Voronoi diagram.

(iv) Update ε_{max}^{j+1} and all residuals associated with tiles adjacent to the new tile τ_{max}^{j+1} with center $\mathbf{p}_{max}$. (All other clusters remain unchanged, *i.e.*, $\tau_i^{j+1} = \tau_i^j$, $\tilde{f}_i^{j+1} = \tilde{f}_i^j$, and $\varepsilon_i^{j+1} = \varepsilon_i^j$.)

(v) Compute the global approximation error ε^j using the error norm (2). Terminate the process when a prescribed global error bound is satisfied or when a prescribed number of points has been inserted. Otherwise, increment j and continue with step (iii).

4 Numerical Results

We have applied the Voronoi-based clustering approach to approximate the terrain data set "Crater Lake", courtesy of U.S. Geological Survey. This data set consists of 159272 samples at full resolution. Approximation results for multiple levels of resolution are shown in Figure 5 and in Table 1.

The quality of approximations obtained with our method compares favorably to results obtained from other multiresolution algorithms like wavelet transforms. A standard compression method, for example, is the use of a wavelet transform followed by quantization and arithmetic coding of the resulting coefficients. Using the Haar-wavelet transform for compression of the Crater-Lake data set (re-sampled on a regular grid at approximately the same resolution) results in approximation errors $(p = 2)$ of 0.89 percent for a 1:10 compression and 4.01 percent for a 1:100 compression [2]. We note that for a Voroni-based compression method also the locations of the samples need to be encoded.

In addition to a hierarchy of smooth approximations, our method provides a cluster hierarchy based on convex cells and an importance ranking for sites. Future work will be directed at the explicit representation of discontinuities and sharp features.

5 Acknowledgements

We thank Mark Duchaineau, Daniel Laney, and the members of the Visualization Group at CIPIC at the University of California, Davis for their helpful ideas and discussions. This work was supported by the DFKI at the University of Kaiserslautern in Germany, Lawrence Livermore National Laboratory (Student Employee Graduate Research Fellowship awarded to the first author), the National Science Foundation under contracts ACI 9624034 and ACI 9983641 (CAREER Awards), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of ALSTOM Schilling Robotics, Chevron, Silicon Graphics, Inc. and ST Microelectronics, Inc.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, Germany, 1997.

[2] M. Bertram, *Multiresolution Modeling for Scientific Visualization*, Ph.D. Thesis, Department of Computer Science, University of California at Davis, 2000. `http://graphics.cs.ucdavis.edu/ bertram`

[3] G. Farin, *Surfaces over Dirichlet tessellations*, Computer Aided Geometric Design, Vol. 7, No. 1–4, 1990, pp 281–292.

[4] L. De Floriani, B. Falcidieno, and C. Pienovi, *A Delaunay-based method for surface approximation*, Proceedings of Eurographics '83, Amsterdam, Netherlands, 1983, pp. 333–350, 401.

[5] L. De Floriani and E. Puppo, *Constrained Delaunay triangulation for multiresolution surface description*, Proceesings Ninth IEEE International Conference on Pattern Recognition, IEEE, 1988, pp. 566–569.

[6] B. Heckel, A.E. Uva, B. Hamann, and K.I. Joy, *Surface reconstruction using adaptive clustering methods*, IEEE Transactions on Visualization and Computer Graphics, submitted, 2000.

[7] B.F.J. Manly, *Multivariate Statistical Methods, A Primer*, second edition, Chapman & Hall, New York, 1994.

[8] A. Maus, *Delaunay triangulation and convex hull of n points in expected linear time*, BIT, Vol. 24, No. 2, pp. 151–163, 1984.

[9] S.E. Schussman, M. Bertram, B. Hamann and K.I. Joy, *Hierarchical data representations based on planar Voronoi diagrams*, R. van Liere, I. Hermann, and W. Ribarsky, eds., Proceedings of VisSym '00, Joint Eurographics and IEEE TCVG Conference on Visualization, Amsterdam, Netherlands, May 2000.

[10] R. Sibson, *locally equiangular triangulation.* The Computer Journal, Vol. 21, No. 2, 1992, pp. 65–70.

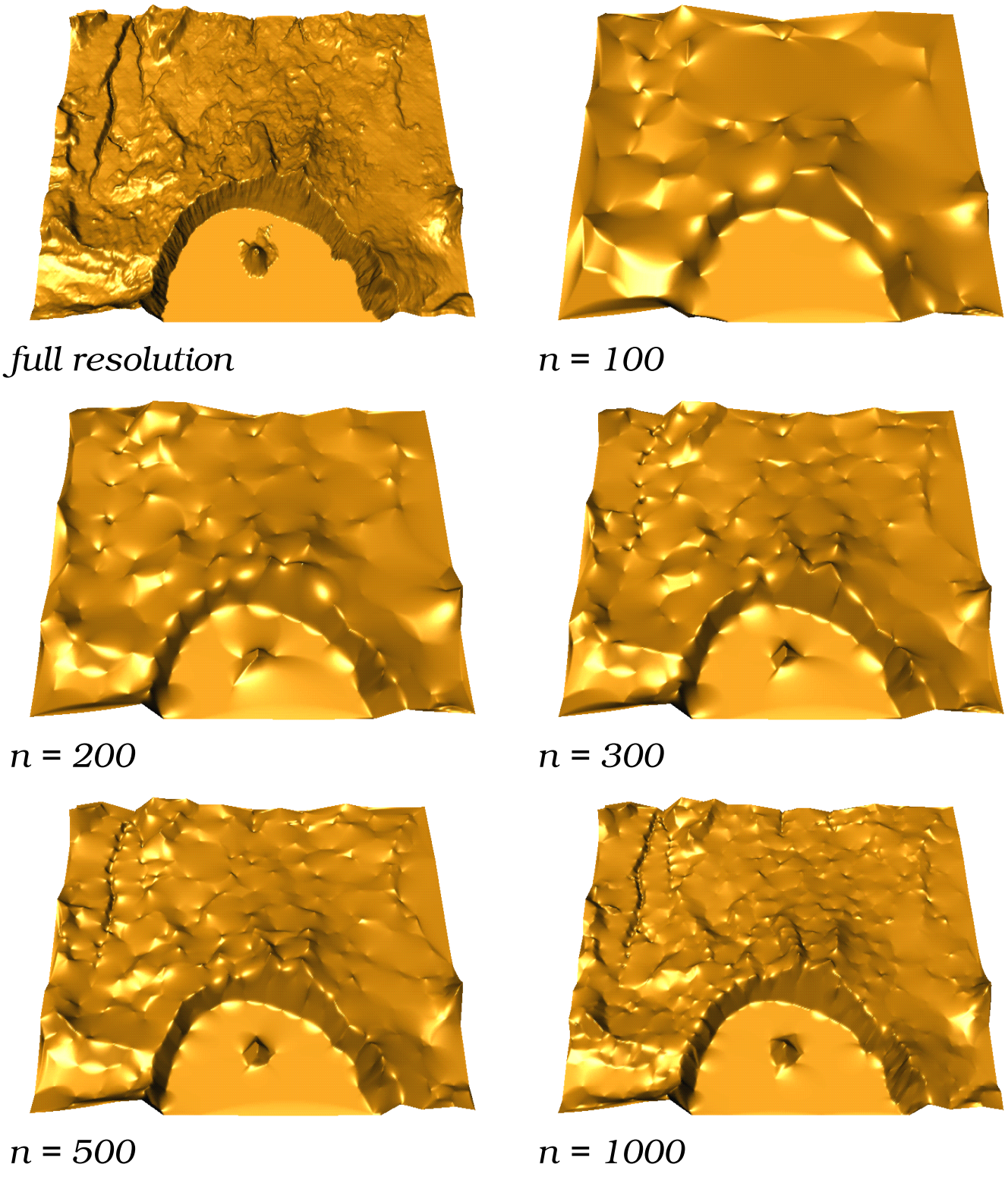


Figure 5: Crater-Lake terrain data set at different levels of resolution.

Simplification of Large, Closed Triangulated Surfaces Using Atomic Envelopes

Peer-Timo Bremer, Oliver Kreylos,
Bernd Hamann, Franz-Erich Wolter

October 2, 2000

1 Introduction

Over the past two decades, data visualization has become increasingly important in several fields, including medical, fluid flow and geographical data. The speed of visualization algorithms has unfortunately not kept up with the speed of developing new technology producing high-resolution data. Every year, the quality of imaging and computational simulation technology — including laser scanners, digital cameras and radar systems — improves substantially. This results in such an increase in the amount of data that even state-of-the-art computers are stretched beyond their capacities. However, it has become apparent that, for many applications large parts of data sets are often not necessary for generating a good picture. The goal was and still is to reduce data sets in such a way that the pictures generated from a reduced data set are highly similar to those produced from the original one.

We are concerned with polygonal surfaces and their compression. Examples for polygonal surfaces are discretized height fields, parametric surfaces, and manifold surfaces. We focus on triangulated two-dimensional (2D) manifolds with no boundaries. For an extensive overview of the field of polygonal surface simplification, we refer to Heckbert and Garland [2] and Rossignac [6].

We present a randomized algorithm that approximates triangulated, orientable 2D manifolds without boundaries, using a "min-#" approach, see §2. The algorithm preserves a specified error bound.

2 Related Work

2.1 Two Approximation Types: Min-ϵ and Min-#

When approximating a polygonal surface using the min-ϵ approach, one has to determine, for a given number n, an approximation that consists of n vertices and minimizes the approximation error. Many of the common algorithms use min-ϵ optimization, and several references are given in [2],[6]. Of special interest is Kreylos and Hamann [4], since they use a method closely related to the one presented here.

Using a min-# approximation approach, one tries to find an approximation with the minimal number of vertices that satisfies a tolerance condition [1]. This approach is relevant for scientific applications. For example, given the size of an object and the view-point distance, one can compute the error tolerance related to one pixel on the screen. Approximating the object within this tolerance results in a picture where each data point is no more than one pixel away from its original location. Computing min-# approximations can be very complicated and expensive. The error metric one wants to minimize is the number of vertices, faces or edges. Additionally, one has to stay inside an error bound. Our algorithm ensures that no point of the approximating surface deviates more than ϵ from the original surface. This requires us to consider an

"offset" around the original surface, and the approximation surface must stay inside this offset. Such an approach was first proposed by Cohen et al. [1] and was, called *simplifiction envelope*. A simplification envelope is a linearized and, in some respects, simplified version of the exact offset.

2.2 Simplification Envelopes

The simplification envelope of a triangulated surface is constructed in the following way: For each vertex, one computes its normal n as a combination of the normals of the surrounding triangles, normalized to length ϵ; one defines two offset vertices, the ($+\epsilon$)-offset and the ($-\epsilon$)-offset vertices, by adding/subtracting n to/from the original vertex. This defines a so-called *fundamental prism*. This approximation of the offset is

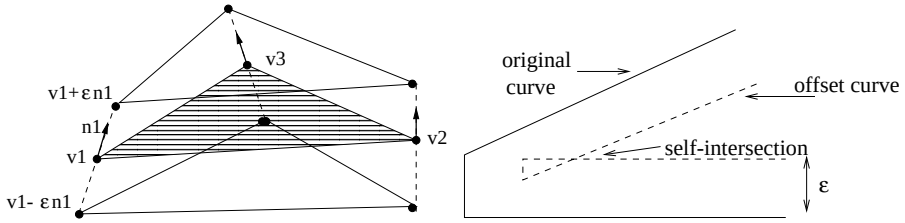


Figure 1: (a) Fundamental prism; (b) self-intersecting offset curves

close to the exact one, as long as the original surface has low curvature. Our approach uses this type of envelope, but it provides the option to use better approximations.

The second problem is caused by self-intersections, see Fig. 1b. Cohen et al. [1] require a simplification envelope that does not self-intersect. They use the global ϵ-value whenever possible and decrease it in areas of possible self-intersections. Our approach is not impacted by self-intersections and can handle every ϵ-value at any given vertex.

3 Atomic Envelopes

To satisfy an a priori error bound, we define *atomic envelopes*. For each triangle, we construct an atomic envelope so that the simplification envelope equals the union of atomic envelopes. Our implementation uses fundamental prisms as atomic envelopes but different constructions are possible when higher accuracy is desired.

During simplification, we have to decide whether a triangle lies inside the simplification envelope. To answer this query we first find all atomic envelopes that might intersect the triangle. Only the top and bottom triangles of these can intersect the triangle in question. We use a bounding box test incorporating an R*-Tree [3] to speed up calculations. Especially for smaller error bounds, this results in roughly the same set of triangles one would get using triangulated offset surfaces to describe the simplification envelope. We intersect all resulting triangles with the triangle being tested. At each resulting intersection point, the triangle might leave the simplification envelope. It leaves the envelope, if and only if the exit point is not covered by another atomic envelope. To test this we use the fact that fundamental prism are pentaeder Bézier volumina [5] and solve the resulting non-linear system of equations.

Cohen et al. [1] define the side faces of a fundamental prism as bilinear patches, defined by the four corner points. Since we deal with closed triangulated surfaces, we do not have to consider the side patches. A triangle cannot leave the envelope through a side patch of a prism: Two neighboring triangles always share one side patch; thus a triangle leaving a prism through a side patch immediately enters another prism.

The same basic algorithm can also be used with more complicated atomic envelopes. An example is the construction shown in Fig. 2. This construction does not only use the vertex normal but also the normal of the triangle to create the atomic envelope. Compared to fundamental prisms, we add three bilinear patches to each atomic

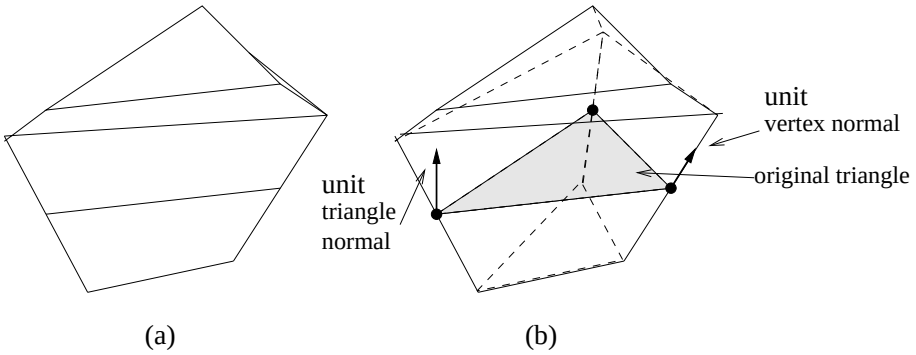


Figure 2: Different atomic envelopes (a) solid view; (b) transparent view

envelope, which can all have possible exit points. Furthermore, to test whether an exit point is covered by an atomic envelope is a consequently slower operation. However, this new atomic envelope approximates the exact non-linear offset much better, especially in regions of high curvature.

4 Simplification

We simplify the given surface using a *simulated annealing* algorithm, also called *Metropolis algorithm*. Simulated annealing models the state transition from fluid to crystalline state of metals. From the algorithmic view-point, this process is an optimization process with extremely high dimension. For our application, we interpret the configuration of a polygonal surface as the configuration of metal molecules. Our internal energy is represented by a target function, and the random heat movement of molecules is represented by random changes in the configuration.

The target function describes the quality of an approximation. Furthermore, the target function should not only prefer configurations that consist of few vertices but also configurations that lead to vertex removals. We use the sum of the square roots of the angles between triangle normals as target function. This function is highly related to the number of vertices. It also prefers planar surfaces, since a large number of small angles has a higher target function value than a smaller number of large angles. This leads to near-planar platelets of triangles, where we can delete vertices. This target function is also easy to compute and can be recomputed locally after local changes.

To change a configuration, we use the method of Kreylos and Hamann [4], adapted to our problem. We use three different operations: edge rotation, vertex removal, and vertex movement. The edge rotation only changes the triangulation of two neighboring triangles. To move a vertex we randomly choose a new position inside a small sphere around the original one. To check the validity of the resulting triangulation we project all involved triangles onto the plane defined by the vertex normal of the changing vertex. This can lead to degenerate triangles or triangles with wrong orientations. We resolve these conflicts by swapping the appropriate edges. To remove a vertex we collapse the shortest edge emanating from it.

5 Future Research

The main drawback of our algorithm is its lack of computational efficiency. Especially the simulated annealing is expensive. However, it provides a mean to use any point inside the envelope as a possible vertex position. Future work will be done to replace the simulated annealing approach, while keeping this advantage.

Figure 3: The original drill bit data set (1964 vertices).

Figure 4: The drill bit data set simplified using 1/4% error bound.

References

[1] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright. Simplification Envelopes. *Proceedings SIGGRAPH 1996* pp. 119-128.

[2] P. S. Heckbert and M. Garland, Survey of polygonal surface simplification algorithms, Multiresolution Surface Modeling Course, SIGGRAPH 1997.

[3] S. Klopp. Implementation von R*-B”aumen als benutzerdefinierte Indexstruktur in Oracle 8i. *Studienarbeit*, Fachbereich Mathematik und Informatik, Universität Hannover 14. Dec. 1999.

[4] O. Kreylos, B. Hamann. On Simulated Annealing and the Construction of Linear Spline Approximations for Scattered Data. *Proceedings of the Joint EUROGRAPHICS-IEEE TVCG Symposium on Visualization*, Vienna, Austria, May 1999, pp. 189-198.

[5] D. Lasser. Bernstein-Bézier-Darstellung trivarianter Splines. *Dissertation*, Fachbereich Mathematik, Technische Hochschule Darmstadt, Germany, 1987.

[6] J. Rossignac. Interactive Exploration of Distributed 3D Databases over the Internet. *SIGGRAPH Proceedings* pp. 324-335, 1998.

[7] Web pages of the Department of Computer Science at Stanford University, http://www.graph- ics.stanford.edu/data/3Dscanrep/

Hierarchical Image-based and Polygon-based Rendering for Large-Scale Visualizations

Chu-Fei Chang* Amitabh Varshney † Qiaode Jeffrey Ge‡

Abstract

Image-based rendering takes advantage of the bounded display resolution to limit the rendering complexity for very large datasets. However, image-based rendering also suffers from several drawbacks that polygon-based rendering does not. These include the inability to change the illumination and material properties of objects, screen-based querying of object-specific properties in databases, and unrestricted viewer movement without visual artifacts such as visibility gaps. View-dependent rendering has emerged as another solution for hierarchical and interactive rendering of large polygon-based visualization datasets. In this paper we study the relative advantages and disadvantages of these approaches to learn how best to combine these competing techniques towards a hierarchical, robust, and hybrid rendering system for large data visualization.

1 Introduction

As the complexity of the 3D graphics datasets has increased, different solutions have been proposed to bridge the growing gap between graphics hardware and the complexity of datasets. Most of algorithms which effectively reduce the geometric complexity and overcome hardware limitations fall into the following categories: visibility determination [7, 9, 2, 1, 8, 6, 10], level-of-detail hierarchies [5], and image-based rendering (IBR) [3]. IBR has emerged as a viable alternative to the conventional 3D geometric rendering, and has been widely used to navigate in virtual environments. It has two major advantages over the problem of increasing of complexity of 3D datasets: (1) The cost of interactively displaying an image is independent of geometric complexity, (2) The display algorithms require minimal computation and deliver real-time performance on workstations and personal computers. Nevertheless, use of IBR raises the following issues:

- Economic and effective sampling of the scene to save storage without visually perceptible artifacts in virtual environments,
- Computing intermediate frames without visual artifacts such as visibility gaps,
- Allowing changes in illumination, and
- Achieving high compression of the IBR samples.

To address some of the above issues we have developed a multi-layer image-based rendering system and a hybrid image- and polygon-based rendering system. We first present a hierarchical, progressive, image-based rendering system. In this system progressive refinement is achieved by displaying a scene at varying resolutions, depending on how much detail of the scene a user can comprehend. Images are stored in a hierarchical manner in a compressed format built on top of the JPEG standard. At run-time, the appropriate level of detail of the image is constructed on-the-fly using real-time decompression, texture mapping, and accumulation buffer. Our hierarchical image compression scheme allows storage of multiple levels in the image hierarchy with minimal storage overhead (typically less than 10%) compared to storing a single set of highest-detail JPEG-encoded images. In addition, our method provides a significant speedup in rendering for interactive sessions (as much as a factor of 6) over a basic image-based rendering system.

We also present a hybrid rendering system that takes advantage of the respective powers of image- and polygon-based rendering for interactive visualization of large-scale datasets. In our approach we sample the scene using image-based rendering ideas. However, instead of storing color values, we store the visible triangles. During pre-processing we analyze per-frame visible triangles and build a compressed data-structure to rapidly access the appropriate visible triangles at run-time. We compare this system with pure image-based, progressive image-based system (outlined above), and pure polygon-based systems. Our hybrid system provides a rendering performance between a pure polygon-based and a multi-level image-based rendering system discussed above. However, it allows several features unique to the polygon-based systems, such as direct querying to the model and changes in lighting and material properties.

2 Multi-Level Image-Based Rendering

In this section, we present an image-based rendering system. This system composes a scene in a hierarchical manner to achieve the progressive refinement by using different resolution images. Progressive refinement is achieved by taking advantage of the fact that the human visual system's ability to perceive details is limited when the relative speed of the object to the viewer is high. We first discuss the pre-processing and then the run-time navigation.

2.1 Image Sampling and Collection

Data sampling and collection plays a very important role in an image-based rendering system. It directly affects the storage space and the real-time performance of the system including image quality, rendering speed and user's visual perception. Different sampling strategies can be applied depending on the purpose of the system.

Environment Setting In our system the model is placed at the center of a virtual sphere.bc 360/ Viewer (camera) is positioned on the sphere with the viewing direction toward the origin. The viewer can move around the sphere along longitude and latitude. The camera takes one snapshot every $\Delta\theta$ degree along longitude and $\Delta\phi$ degree along latitude. Due to the symmetry of sphere, we will have $360/\Delta\theta \times 180/\Delta\phi$ camera positions. The sampling density of camera positions may be adjusted by changing the values of $\Delta\theta$ and $\Delta\phi$. In our implementation, $\Delta\theta = \Delta\phi = 5^\circ$, to achieve a reasonably smooth and continuous motion with 2592 images.

*Department of Applied Mathematics, State University of New York, Stony Brook, NY 11794, chchang@cs.sunysb.edu

†Department of Computer Science, University of Maryland, College Park, MD 20742, varshney@cs.umd.edu

‡Department of Mechanical Engineering, State University of New York, Stony Brook, NY 11794, ge@design.eng.sunysb.edu

2.2 Multi-Level Image Construction Algorithm

The algorithm computes n different levels of resolutions of images as the basis for building the system image database. Our algorithm has the following steps:

Step 1: Decide the number n of progressive refinement levels in the system and the resolution of the display window, say $W \times W$, where $W = 2^m$, and $m \leq n$.

Step 2: Dump a Level 0 image, say I_0, at the display window resolution ($W \times W$).

Step 3: Construct Level $i+1$ image (resolution $= W/2^{i+1} \times W/2^{i+1}$), say I_{i+1}. The RGB values of level $i+1$ image are constructed from the RGB values of level i image by the following equations:

$$R^{i+1}_{j,k} = min\{R^i_{2j,2k}, R^i_{2j+1,2k}, R^i_{2j,2k+1}, R^i_{2j+1,2k+1}\} \quad (1)$$

$$G^{i+1}_{j,k} = min\{G^i_{2j,2k}, G^i_{2j+1,2k}, G^i_{2j,2k+1}, G^i_{2j+1,2k+1}\} \quad (2)$$

$$B^{i+1}_{j,k} = min\{B^i_{2j,2k}, B^i_{2j+1,2k}, B^i_{2j,2k+1}, B^i_{2j+1,2k+1}\} \quad (3)$$

where $i = 0, 1, \ldots, n-2$. For example, R^{i+1}_{00} is computed by $min\{R^i_{00},\ R^i_{10},\ R^i_{01},\ R^i_{11}\}$. We repeat this step until I_{n-1} image is computed.

Step 4: Compute $W/2^i \times W/2^i$ resolution image T_i from $W/2^{i+1} \times W/2^{i+1}$ resolution image I_{i+1} as follows. Display I_{i+1} on a $W/2^i \times W/2^i$ resolution window using texture mapping and dump the displayed window image as T_i. Compute image difference D_i as:

$D_i = I_i - T_i, \quad i = 0, 1, \ldots, n-2$

Repeat this step until D_{n-2} image difference is computed, see Figure 1.

Step 5: Store I_{n-1}, D_{n-2}, D_{n-3}, ..., D_0 in JPEG format as the database images.

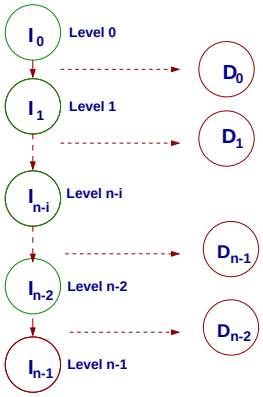


Figure 1: Multi-Level Image Construction

This algorithm works well since texture mapping hardware provides speed and antialiasing capabilities over OpenGL function `glDrawPixels()`. Also, image differences compress better than full images and provide an easy way to generate progressive refinement. For compression and decompression we use the public-domain JPEG software [4] in our implementation. It supports sequential and progressive compression modes, and is reliable, portable, and fast enough for our purposes. The reason we take the minimum value in equations 1–3 is so that we can store all RGB values of D_i as positive values and save a sign bit in storage.

2.3 Progressive Refinement Display Algorithm

Let us define $Tex(I_{n-1})$ as the image by texture mapping image I_{n-1} on $W \times W$ resolution window, and define $Tex(D_i)$ as the image by texture mapping image D_i on $W \times W$ resolution window, where $i = 0, 1, \ldots, n-2$. At run time, level i image is displayed by accumulating images $Tex(I_{n-1})$, $Tex(D_{n-2})$, $Tex(D_{n-3})$, ..., $Tex(D_i)$, where $i = 0, 1, \ldots, n-1$. If $i = n-1$, we only display image $Tex(I_{n-1})$, which has the lowest detail. We add $Tex(D_i)$ image onto $Tex(I_{n-1})$, where $i = n-2, n-3, \ldots, 0$, to increase the image details. Level 0 image, which is $Tex(I_{n-1}) + \sum_{i=0}^{n-2} Tex(D_i)$, has the highest detail. Notice that all images are decompressed before texture mapping. The implementation is done by OpenGL accumulation buffer and texture mapping.

In a real-time environment, the progressive refinement can be achieved by displaying different levels of images, depending on how much detail of the scene the user needs to see. If the user moves with high speed, we can simply display lowest detail. As the user speed reduces, we can raise the level of detail of the displayed image in a progressive fashion.

2.4 Our Implementation and Results

In our implementation, we use three different image resolutions, 128×128, 256×256, and 512×512, for progressive refinement. We use sequential mode with quality setting 80 for JPEG compression, which gives us an unnoticeable difference from the highest quality setting of 100. We observed that the composite image quality in our system is not only affected by the lossy JPEG compression, but also by the error from image difference and the geometric error from texture mapping. Table 1 shows the JPEG image reduction from full images I to image differences D. $\sum I_i$ is the sum of storage for all the I_i (level i) images. Similarly, $\sum D_i$ is the sum of storage for all the D_i (level i) images. The total storage is computed as $\sum(I_2 + D_1 + D_0)$. Note that the total storage compares quite favorably to original (non-progressive) storage requirements ($\sum I_0$).

Model	Level 2 $\sum I_2$	Level 1 $\sum I_1$	Level 1 $\rightarrow \sum D_1$	Level 0 $\sum I_0$	Level 0 $\rightarrow \sum D_0$	Total (MB)
Bunny	10.70	21.39	10.70	53.11	31.34	52.74
Submar.	19.35	40.54	29.63	112.53	70.13	119.11
EHydr.	21.29	37.88	21.31	107.34	46.26	88.86
Dragon	12.19	25.73	21.15	64.70	42.61	75.95
Buddha	10.70	20.15	14.22	47.22	30.86	55.78

Table 1: Storage for I_i, D_i and the total system

Image Level	Decompression Time (msec)	Rendering Time (msec)	Speed (fps)	Image Error
$I_2 + D_1 + D_0$	98.6	23.8	7.99	0.435
$I_2 + D_0$	25.4	16.6	23.80	0.077
I_2	6.3	9.4	63.96	0.079
I_1	20.9	10.1	32.31	0.025
I_0	78.9	17.4	10.37	0.0

Table 2: Multi-Level Image Rendering Comparison

Table 2 shows the decompression time, rendering time, and frame rate on different image levels. All numbers in this table are the average numbers over different models. I_i, $i = 0, 1, 2$ are full images of 512×512, 256×256, and 128×128 resolutions, respectively. D_i, $i = 0, 1$ are the image differences we discussed in

Section 2.2. The *image error* is the root-mean-squared difference between the two images. The errors reported are with respect to the I_0 image.

3 Hybrid Rendering

Image-based rendering is a two-stage process. The first stage is off-line preprocessing that includes sampling of the necessary scene information and setting up data structures, possibly with hierarchy and compression, to reduce access times. The second stage deals with real-time rendering of pre-processed image data which may include image interpolation and warping. Like conventional image-based method, our hybrid method also has two stages and the key difference is that, instead of using three- or four-channel color values for each image, we compute the exact visibility of each triangle for each viewpoint, and only the visible (displayed) triangles are stored for each viewpoint.

3.1 Preprocessing

We adopt the same environment settings as we did in the JPEG image-based rendering system, see section 2.1.

3.1.1 Encoding Triangle IDs

In order to compute the visibility for each triangle, we assign each triangle a unique id when we load the dataset. We then decompose the number, in binary format, into three consecutive bytes and assign them to R, G, and B in order. During the dumping process, we render the whole dataset with the given RGB value for each triangle as its color. Notice here that in order to render all colors correctly, the illumination and antialiasing function in OpenGL should be turned off. We then read the color buffer of this image to get the color for each pixel and compose the R, G, B back to the id. We currently use unsigned char for each single color value, which means, with a one-pass encoding process, we can encode as many as $(2^8)^3 = 16$ million triangles. For larger datasets, multiple-pass encoding processes may be needed. In our method we dump triangles for each camera position (θ, ϕ) by using the dumping process we discussed in Section 2.1 into a occupancy bit-vector, say TriMap(θ, ϕ).

3.1.2 Compression Process

Two types of compression are relevant in an image-based navigation of virtual environments: single-frame compression and frame-to-frame compression. We have only worked with single frame compression at this stage; the multiple frame compression, which needs more analysis and work, will be dealt with in future. For representing the visible triangles in a single frame we use an occupancy bit vector (an unsigned char array) in which each bit represents the triangle id corresponding to its position in the vector. The bit is 1 is the triangle is visible in that frame, 0 otherwise.

As the size of 3D datasets increases and the resolution of image space remains fixed, the number of dumped triangles will saturate around the display resolution. In our results, the million triangle Buddha model has on an average only $5 \sim 6\%$ visible triangles for a 512×512 resolution window. It means that most bits in a bit vector would be 0 and consecutive-0-bit-segment cases occur frequently. This result inspires us to use run-length encoding and Huffman compression.

3.2 Run-time Navigation

At run time the 3D dataset and precomputed information in compressed format is loaded first. The precomputed information not only includes the visible primitives for each frame but also includes the viewing parameters including viewing angle, distance, and so forth. The run-time viewing parameters should be exactly the same as those used in the dumping process. In the system, each camera position has a frame pointer pointing to the corresponding frame in compressed format. A Huffman tree, which is used for decompression, is also constructed for each frame.

At run time the viewer moves around in a virtual environment following discrete camera positions at $\Delta\theta$, $\Delta\phi$ increments which were used in the dumping process. For a given viewer positon, we can locate the corresponding frame by following its frame pointer and decompress the frame by retracing the Huffman tree.

The rendering speed of system highly depends on the number of visible triangles and the decompression time. In our implementation, the decompression function doesn't have to go through a whole data frame, it breaks the decompression loop immediately whenever it detects that all dumped triangles have been found and sends them to the graphics engine. However, the decompression time still depends on the size of the frame (the size of object model) and the number of visible triangles in the frame.

4 Results

We have tested five different polygonal models on SGI Challenge and Onyx2. All models are tested on 512×512 resolution window with 2592 images. We describe our results in this section.

Bunny, Dragon, and Buddha are scanned models from range images from the Stanford Computer Graphics Lab. Submarine model is a representation of a notional submarine from the Electric Boat Division of General Dynamics. The E. Hydratase Molecule (Enoyl-CoA Hydratase) is from the Protein Data Bank. All models have the vertex coordinates (x, y, z) in floating-point format, and all triangles are represented by their three vertex indices (integer). Submarine has RGB color for values (unsigned char) for each triangle. The E. Hydratase Molecule has a normal vector for each vertex. All models are stored in OFF binary format.

Table 3 has the average compression ratios on all models. The *average dumped tris %* is the average percentage of dumped triangles over 2592 images.

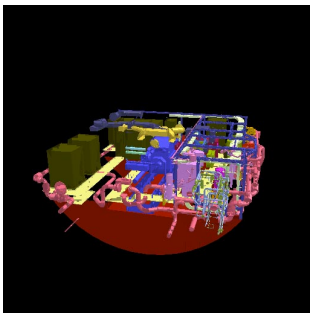
Model	Avg Dumped Tris %	Run Length Ratio	Huffman Ratio	Total Ratio
Bunny	35.68 %	1.47	1.29	1.82
Submarine	2.83 %	6.27	1.46	9.16
E. Hydratase	11.21 %	3.54	1.24	4.38
Dragon	7.79 %	1.68	1.60	2.69
Buddha	4.04 %	2.32	1.75	4.07

Table 3: Compression Ratios for the Hybrid Method

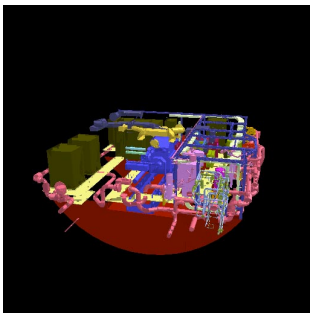
In Table 4, *P* refers to conventional *Polygonal* rendering, *H* refers to the *Hybrid* rendering system discussed in Section 3, *I* refers to the *Multi-level Image-based* rendering system discussed in Section 2. The *Image Error* is the root-mean square error with respect to the images rendered from the conventional polygonal rendering. *Dcmprs* is the time for decompression. As can be seen, the *Hybrid* method has consistently low image errors. Multi-level image-based rendering has the highest image error amongst these methods, since JPEG compression, image differences, and texture mapping all contribute to the final image error. For the *Hybrid* method, all visible triangles are stored in a bit vector and compressed by two steps: run length encoding and Huffman compression. The decompression and rendering speeds are highly dependent on the displayed frame size and the number of dumped triangles in that frame. The rendering speed on the Submarine is much slower than other models

because we do the coloring for each rendered triangle. Without coloring, the *Polygonal* method has the average rendering speed about $7-9$ frames/sec and the *Hybrid* method is over 10 frame/sec.

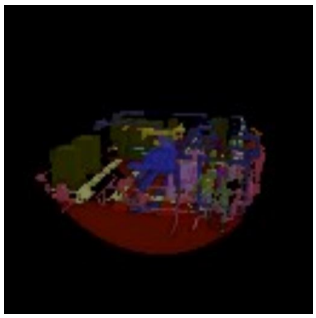
The traditional polygon rendering has the best quality amongst all methods and least storage, but it has the lowest rendering speed. The hybrid method which only renders visible triangles has very good rendering speeds, but needs much more storage than traditional polygon rendering. Multi-level JPEG provides progressive refinement and has the lowest rendering complexity, but needs a lot of storage. Figure 2 shows the images displayed by these methods on various models.



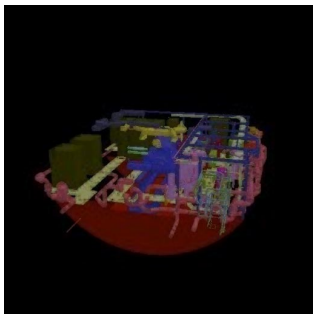
(a) Polygonal: 376,436 tris



(b) Hybrid: 5,201 tris



(c) JPEG Level 0



(d) JPEG Level 2

Figure 2: Different Rendering Methods on the Submarine

Model	System	Storage MB	Time and Speed			Image Error
			Dcmprs (msec)	Render (msec)	Speed (fps)	
Bunny 69K tris	P	1.54	0.0	61.3	16.39	0.0
	H	12.35	10.8	81.7	10.79	2.02E-4
	I	52.74	83.6	28.2	8.94	2.66E-2
Submarine 376K tris	P	12.85	0.0	3549.0	0.28	0.0
	H	13.32	11.1	118.1	7.73	7.29E-3
	I	136.20	107.8	27.1	7.40	1.42E-1
EnoylCOA Hydratase 717K tris	P	14.95	0.0	777.4	1.28	0.0
	H	37.93	32.1	177.8	4.76	4.15E-4
	I	88.86	108.6	28.9	7.26	2.69E-2
Dragon 871K tris	P	19.19	0.0	1306.9	0.76	0.0
	H	104.98	87.8	223.45	3.10	4.82E-4
	I	75.95	102.2	28.7	7.63	2.45E-3
Buddha 1087K tris	P	23.92	0.0	1638.3	0.61	0.0
	H	86.56	69.4	191.9	3.82	5.14E-4
	I	55.78	95.9	27.9	8.07	9.60E-2

Table 4: Comparison Results for Different Methods

5 Conclusions

In this paper we have presented a hybrid method as well as a progressive refinement image-difference-based rendering method for high-complexity rendering. Our hybrid method takes advantage of both conventional polygon-based rendering and image-based rendering. The hybrid rendering method can provide rendering quality comparable to the conventional polygonal rendering at a fraction of the computational cost and has storage that is comparable to the image-based rendering methods. The drawback is that it does not permit full navigation capability to the user as in the conventional polygonal method. However, it still retains several other useful features of the polygonal methods such as direct querying to the underlying database and ability to change illumination and material properties. In future we plan to further explore compression issues for the hybrid method by taking advantage of frame-to-frame coherence in image space and view-dependent geometric hierarchical structures.

6 Acknowledgements

This work has been supported in part by the NSF grants: DMI-9800690, ACR-9812572, and IIS-0081847.

References

[1] Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, pages 1–7, June 1998.

[2] S. Coorg and S. Teller. Real time occlusion culling for models with large occluders. In *Proceedings of 1997 Simposium in 3D Interactive Graphics*, pages 83–90, 1997.

[3] P. Debevec, C. Bregler, M. Cohen, L. McMillan, F. Sillion, and R. Szeliski. *SIGGRAPH 2000 Course 35: Image-based Modeling, Rendering, and Lighting*. ACM SIGGRAPH, 2000.

[4] Independent JPEG Group. ftp://ftp.uu.net/graphics/jpeg/.

[5] D. Luebke, J. Cohen, M. Reddy, A. Varshney, and B. Watson. *SIGGRAPH 2000 Course 41: Advanced Issues in Level of Detail*. ACM SIGGRAPH, 2000.

[6] M. Panne and A.J. Stewart. Effective compression techniques for precomputed visibility. In *Springer Computer Science*, Rendering Techniques '99, pages 305–316, 1999.

[7] S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics Proceedings(SIGGRAPH 91)*, pages 61–69, 1991.

[8] Y. Wang, H. Bao, and Q. Peng. Accelerated walkthroughts of virtual environments based on visibility preprocessing and simplification. In *Eurographics*, pages 17(3): 187–194, 1998.

[9] R. Yagel and W. Ray. Visibility computation for efficient walkthroughs of complex environments. In *Presence*, pages 5(1):45–60, 1995.

[10] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, Computer Graphics Proccedings, Annual Conference Series, pages 77–88. ACM SIGGRAPH, ACM Press, August 1997.

Volume Visualization of Large Tetrahedral Meshes on Low Cost Platforms

Paolo Cignoni[1], Leila De Floriani[2], Paola Magillo[2], Enrico Puppo[2], Roberto Scopigno[3]

[1] Istituto di Elaborazione dell'Informazione – Consiglio Nazionale delle Ricerche*

[2] Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova†

[3] Istituto CNUCE – Consiglio Nazionale delle Ricerche‡

Abstract

In this paper, we present an interactive system for visualization of three-dimensional scalar fields. The system has been designed in order to overcome some of the problems posed by very large volume data sets. The adopted solution exploits an efficient multiresolution data structure based on a tetrahedral domain decomposition. The mesh to be rendered can be selectively refined over areas that the user considers more critical, on the basis of either field values or domain locations. The system supports different rendering techniques, such as isosurface fitting and Direct Volume Rendering (DVR).

1 Introduction

Space requirements and rendering time for a tetrahedral mesh are proportional to its resolution, i.e., to the density of its tetrahedra, and, thus, to the density of the underlying point set. When a tetrahedral mesh is too large, data simplification can be used in order to bring it to a more manageable size prior to visualization [3, 1, 8, 7, 6, 11].

In many cases, it is very useful to apply simplification selectively, e.g., only in a portion of the domain, or only in the proximity of interesting field values. This process is usually called *selective refinement*.

Mesh simplification, however, is too time-consuming to be performed on line. A more recent approach consists of decoupling the simplification phase from the selective refinement phase. This has led to multiresolution models, which have been developed in the two-dimensional case for triangle meshes (see, e.g., a survey in [5]). Basically, such models encode the simplification steps as a partial order, from which a virtually continuous set of meshes at different Levels Of Detail (LODs) can be extracted, based on efficient graph traversal strategies.

Here, we present a volume visualization system, called *TAn2* (*Tetrahedra Analyzer* 2), which is based on a multiresolution model which specializes for the three dimensional case the model proposed in [4, 9], called the *Multi-Tesselation* (MT).

*CNR Research Area – S. Cataldo 56100 Pisa, ITALY – Email: cignoni@iei.pi.cnr.it

†Via Dodecaneso, 35, 16146 Genova, ITALY – Email: {deflo,magillo,puppo}@disi.unige.it

‡CNR Research Area – S. Cataldo 56100 Pisa, ITALY – Email: roberto.scopigno@cnuce.cnr.it

TAn2 exploits multiresolution not only for reducing data complexity uniformly, but also for varying the resolution through data domain. Selective refinement can be based either on the levels of the isosurfaces to be rendered, or on a spatial location within the domain. These features allow the user to focus on those areas that are considered more critical.

One of the major objectives of *TAn2* is to enable a low cost PC platform to perform those tasks that are nowadays the realm of more costly graphics workstations. We assume a PC architecture equipped with 512MB RAM and a graphics board having a sustained throughput at least of 1M shaded triangles/sec. We consider an application which manages datasets based on a *tetrahedral decomposition* of the 3D space, with scalar data associated with the mesh geometry, and supports Direct volume Rendering (DVR), isosurface fitting and cross-slice rendering at a reasonable degree of interactivity (10 fps). Under the above assumptions, a standard volume visualization system could only work with a mesh not exceeding either 350K vertices when doing isosurface rendering, or 3K vertices when doing DVR (see [2] for details). On the contrary, *TAn2* can manage meshes of up to 12M vertices by using user-driven selective refinement prior to visualization. A more detailed description, presenting also the underlying data structure and the selective refinement techniques on which *TAn2* is based, can be found in [2].

2 The *TAn2* system

The architecture of the *TAn2* (Tetrahedra Analyzer 2) system has been designed to overcome some of the problems arising interactive visualizing very large volume data sets. The solution adopted in *TAn2* is based on a multiresolution structure that describes the whole data set in an implicit and very compact way, and supports selective refinement efficiently [2]. From such structure, smaller subsets of data are extracted and rendered. The amount of data extracted for rendering and their level of resolution (possibly variable over the domain) are selected in such a way to give the largest mesh that can be rendered under the given time and resource constraints. The system has been implemented in C++ and runs under both Windows and Linux.

TAn2 system provides both visualization through isosurfaces, cross sections and Direct Volume Rendering (DVR) on *uniform* or *variable LOD*. In isosurface rendering mode, one or more isosurfaces may be displayed

at a time; data visualized can be enriched by introducing cross section planes (see Figure 3). The DVR approach adopted is based on a depth-sorting and a composition of tetrahedral cells [10, 12].

In all rendering modalities (isosurfaces, cross planes and DVR), appearance parameters such as colors and transparency values are determined by means of a user-defined *Transfer Function* (TF). The Transfer Function defines how field values are mapped into color and transparency values that will be used in rendering. The TF is defined in terms of RGBA components through linear interpolation of the RGBA values specified by the user at a finite set of field values. The system contains a visual interactive tool for editing transfer functions (see Figure 2).

2.1 Role of Multiresolution within the System

Within the system, a compact *multiresolution data structure*, the Multi-Tesselation (MT), is maintained, which describes the whole data set in an implicit way. This data structure requires only about $36n$ bytes for storing a model having n vertices at the highest resolution, and for supporting all multiresolution operations. In [2], we show that the MT has a compression factor of about 6 with respect to storing juat the mesh at the maximum resolution with a standard data structure.

For the purpose of rendering, a *current mesh* is stored, which is dynamically extracted from the MT by selecting just a subset of the data. Rendering is performed based on the current mesh. The current mesh has a suitable size to be rendered with the available memory and with an interactive frame rate, depending on the current viewing modality. The resolution of the current mesh is either uniform or variable over the domain, and is chosen based on the current viewing parameters so as to provide the highest possible quality of displayed images within the size constraints. In particular, the system supports the following uses of multiresolution:

- *Uniform LOD*: the resolution of the current mesh used in rendering is uniform over the domain based on an approximation error threshold specified by the user.

- *Variable LOD based on spatial location in the domain*: the user specifies both an error threshold and a focus volume, e.g., an axis-aligned box. The focus volume is used as a sort of 3D magnifying glass in order to explore a dataset by increasing resolution only locally.

- *Variable LOD based on field value*: the current mesh satisfies the error threshold specified by the user just in proximity of the isosurface values currently selected, while the resolution of the mesh in areas that do not contain such values is coarser.

TAn2 accepts as input either a plain tetrahedral mesh or a Multi-Tesselation. The construction of the Multi-Tesselation is performed off-line by an independent application; we use a simplification code based on edge collapse described in [1].

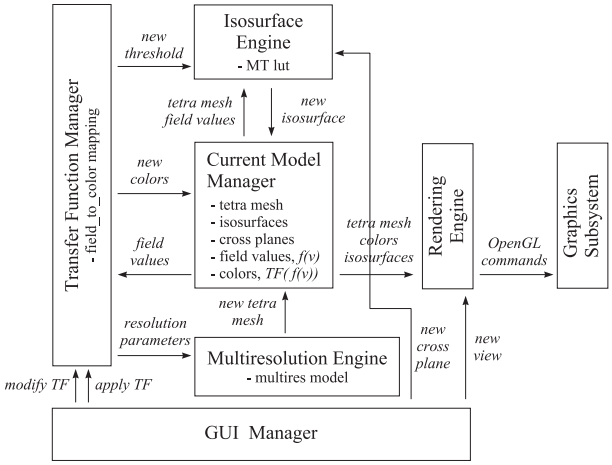


Figure 1: The architecture of the *TAn2* system.

3 System Architecture

The architecture of the *TAn2* system is depicted in Figure 1 and consists of six modules which are described in the following paragraphs. The **GUI Manager** is responsible for rendering all GUI-related areas of the system interface, managing user interaction, and filtering all user inputs before passing them to other modules.

The interface of *TAn2*, shown in Figure 2, consists of one main window subdivided into two regions. The *View region* (on the left) is devoted to the visualization of the dataset. The 3D scene visualized in the View region is managed by the *Rendering Engine*. The *Transfer Function region* (on the right) gives an innovative unified GUI framework for the interactive setting of the current TF, the selection of isosurface thresholds, and the selection of the error threshold for mesh extraction from the MT.

The *Transfer Function region* contains a visual editor to interactively design or modify the current TF, to set up the parameters for the selection of isosurfaces and for the extraction of the current mesh from the MT. Changes to either the TF, or the isosurface values, or the error threshold, are managed by the *Transfer Function Manager* with the help of the *Multiresolution Engine* and of the *Current Model Manager*.
The **Transfer Function Manager** collects all parameters set by the user, which are related to the current TF, to the current isosurface values, and to the current error threshold. The Transfer Function Manager acts as a filter that redirects the user specifications to the appropriate modules which handle them.
The **Multiresolution Engine** is driven by interactively manipulating either the handle of the current error threshold in the Transfer Function region of the GUI, or the *manipulator* 3D widget provided to select the spatial focus region. The *GUI Manager* communicates any changes to the *Transfer Function Manager*, which in turn invokes the *Multiresolution Engine*. Based on the new error threshold, the *Multiresolution Engine* extracts a new mesh from the MT and sends it to the *Current Model Manager*.

The **Current Model Manager** is devoted to the management of the current mesh: a dynamically modifiable tetrahedral mesh that stores the model as extracted from the MT, enriched with data needed in various visualization modalities (like isosurfaces, colors, etc), in other words, it maintains the *ready-to-render* model.
The **Isosurface Engine** computes new isosurfaces on command of the *Transfer Function Manager*, or new cross sections following the requests of the *GUI Manager*. These updated surface meshes are maintained in the *Current Model Manager*.
The **Rendering Engine** performs rendering either of isosurfaces, or directly of the data set (possibly through DVR).

In the example of Figure 2, a uniform resolution mesh has been extracted (the corresponding error threshold is the one selected with the multiresolution extraction handle), and three different isosurfaces are fitted and rendered (corresponding to the three isosurface handles shown) together with the mesh boundary surface.

4 Results

In this section, we show images and performance statistics obtained from the *TAn2* system on the **Turbine Blade** dataset (106,795 vertexes, 576,576 tetrahedra), a curvilinear mesh courtesy of AVS Inc. (tetrahedralized by O.G. Staadt). Turbine Blade is a rather large dataset, non-convex, containing many degenerate cells (i.e., having zero volume). It contains 41,920 boundary cells. An estimate of the average size of the isosurfaces fitted on this data set has been computed by considering nine field values equally distributed in the dataset field range: the estimated average size is about 70K faces per isosurface.

Meshes at uniform resolution have been extracted from the multiresolution representation of the Turbine dataset according to a constant error threshold. The sizes of such meshes are reported in Table 1, where the error value is measured as a percentage of the field range. Table 1 contains also results on isosurface generation from *variable LOD* meshes constructed on the basis of the field value. The value of the resolution threshold used in mesh extraction ($\varepsilon = 0., 0.1$ and 1.0) are measured as percentages of the dataset field range; the corresponding mesh sizes are also reported. It has to be noted that if the dataset contains an isosurface which intersects a large number of mesh cells, then the corresponding *variable LOD* cannot be very concise. An example is provided by the *variable LOD* mesh corresponding to the active isosurface of value 160.28 and to an error $\varepsilon = 0.$ (see Table 1). In this case, the joint use of a *field*-based and of a *spatial domain*-based focusing filter can improve data reduction capabilities. While under the configuration described above ($\varepsilon = 0.$, isosurface=160.28) we obtain a *variable LOD* mesh composed of 473K cells, we can easily produce a representation of smaller size (in the range of 80K cells) by focusing on a domain subvolume. The multiresolution extraction time is in general nearly interactive.

Finally, images representing the other two rendering modalities are in Figures 3 and 5. In the first one, we show a cross plane extracted using a *variable LOD* (40k cells out of the original 576k) under a resolution error threshold $\varepsilon = 2\%$. In the latter, we show a hybrid DVR image (integrated isosurfaces plus DVR) on the same *variable LOD*.

5 Conclusions

The *TAn2* system supports rendering of datasets that are about six times larger than those that could fit in memory with a traditional system, and from 30 to 4000 times larger than those that could be visualized interactively (depending on whether we are doing isosurface rendering or DVR).

To our knowledge, *TAn2* is the only multiresolution volume visualization system able to deal with irregular datasets and non-convex domains efficiently.

References

[1] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral volume with accurate error evaluation. In *Proceedings IEEE Visualization 2000*, page in press. ACM Press, October 7-13 2000.

[2] P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. *TAn2* - visualization of large irregular volume datasets. Technical Report DISI-TR-00-07, DISI, University of Genova (Italy), 2000. (submitted paper).

[3] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and rendering of volume data based on simplicial complexes. In *Proceedings 1994 Symposium on Volume Visualization*, pages 19–26. ACM Press, October 17-18 1994.

[4] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97*, pages 103–110, Phoenix, AZ (USA), October 1997.

[5] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Eurographics '99 – State of the Art Reports*, pages 111–131, 1999.

[6] M.H. Gross and O.G. Staadt. Progressive tetrahedralizations. In *Proceedings IEEE Visualization'98*, pages 397–402, Research Triangle Park, NC, 1998. IEEE Comp. Soc. Press.

[7] R. Grosso and G. Greiner. Hierarchical meshes for volume data. In *Proceedings CGI'98*, Hannover, Germany, June 22-26 1998.

[8] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 217–224, 1997.

[9] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry*, pages 202–210, Ottawa, Canada, August 12-15 1996.

[10] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):63–70, November 1990.

[11] I.J. Trotts, B. Hamann, and K.I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.

[12] P.L. Williams. Visibility ordering of meshed polyhedra. *ACM Transaction on Graphics*, 11(2):103–126, April 1992.

Turbine variable resolution models										uniform resolution models	
isosurface threshold	ε	tetra	isosurf. size	ε	tetra	isosurf. size	ε	tetra	isosurf. size	ε % of field range	
137.38	0.	352,556	73,952	0.1	134,901	33,759	1.	40,611	16,296	0.1	233,848
160.28	0.	473,625	141,978	0.1	169,824	67,909	1.	46,048	24,017	0.5	117,696
183.17	0.	263,311	28,718	0.1	100,798	18,081	1.	36,148	8,343	1.0	84,743
206.07	0.	190,988	8,296	0.1	69,604	6,858	1.	28,637	3,167	5.0	37,515
217.52	0.	108,298	3,001	0.1	47,810	2,621	1.	22,486	1,548	10.0	23,779

Table 1: Results of generating isosurfaces through the extraction of meshes at *variable* resolution from the multiresolution representation of the Turbine dataset. The error is measured as a percentage of the field range, and depends on the current selected isosurfaces (it is $\leq \varepsilon$ on tetrahedra contributing to the isosurface cells; any error on other tetrahedra). Last two columns show sizes of meshes at *uniform* resolution extracted from the multiresolution representation of the Turbine dataset.

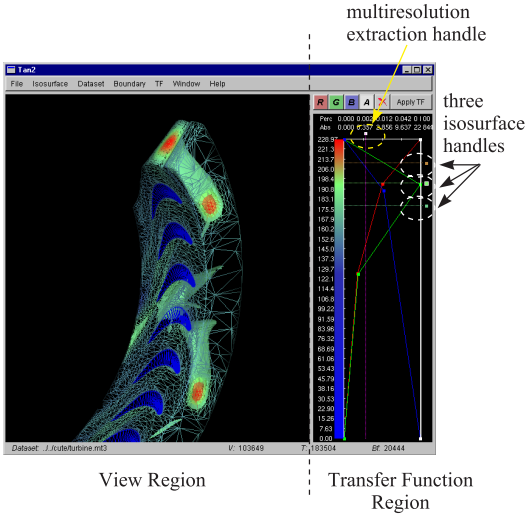


Figure 2: The main window of the *TAn2* system. The isosurfaces shown have been computed on a *uniform LOD* mesh (containing 183K tetrahedra) extracted from the multiresolution representation of the Turbine dataset.

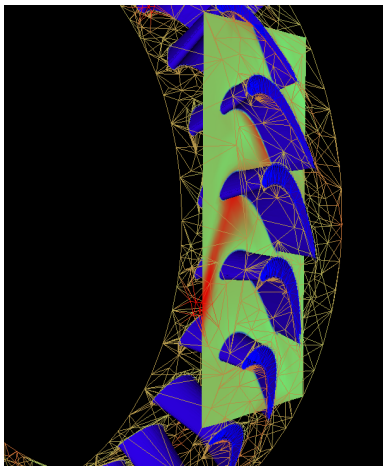


Figure 3: An image of a Turbine *variable LOD* (40k cells, $\varepsilon = 2\%$) with a cross plane.

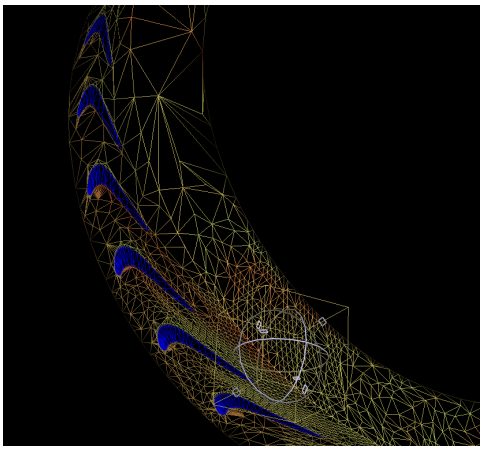


Figure 4: An image of a Turbine *variable LOD* mesh (65k cells, extracted under approximation error $\varepsilon = 0.01\%$ in the interior of the focus region, and error $\varepsilon = 10\%$ in the external space).

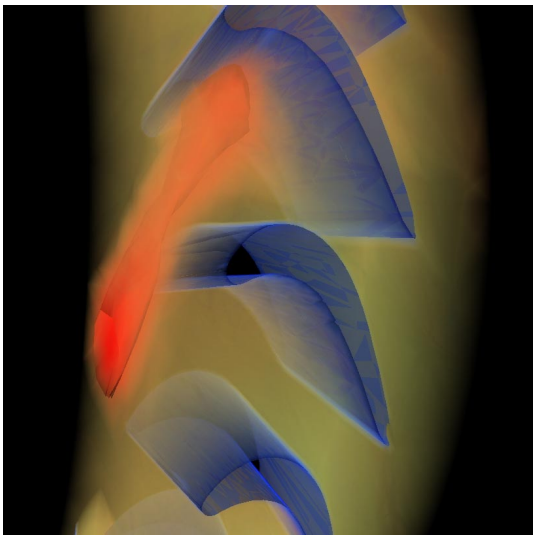


Figure 5: An hybrid DVR (integrated isosurfaces plus DVR) image of a Turbine *variable LOD* (40k cells, $\varepsilon = 2\%$) is shown.

Dataflow and Remapping for Wavelet Compression and Realtime View-Dependent Optimization of Billion-Triangle Isosurfaces

Mark A. Duchaineau[1] Serban D. Porumbescu[1,3] Martin Bertram[2] Bernd Hamann[3] Kenneth I. Joy[3]

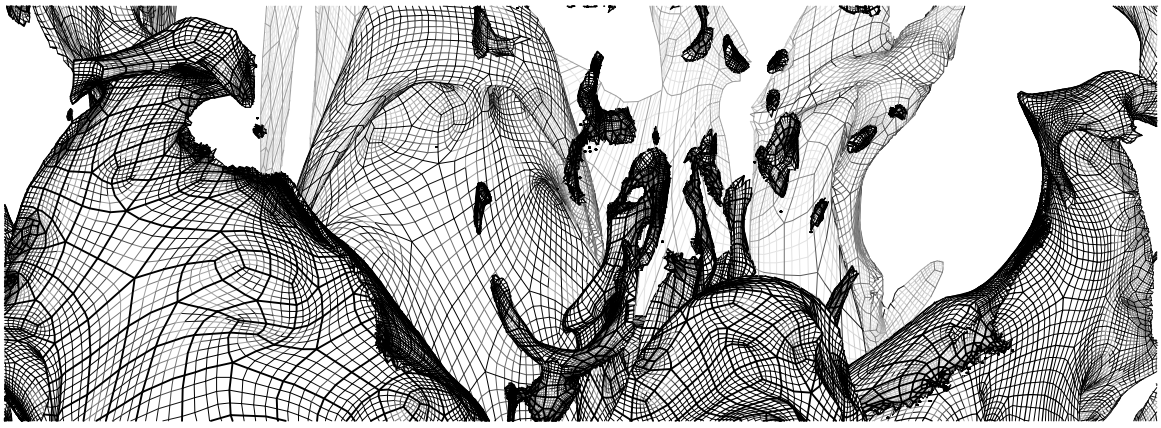


Figure 1: Shrink-wrap result for .3% of a terascale-simulation surface. This conversion from irregular mesh to a semi-regular form enables wavelet compression and view-dependent optimization.

Abstract

Currently, large physics simulations produce 3D fields whose individual surfaces, after conventional extraction processes, contain upwards of hundreds of millions of triangles. Detailed interactive viewing of these surfaces requires powerful compression to minimize storage, and fast view-dependent optimization of display triangulations to drive high-performance graphics hardware. In this work we provide an overview of an end-to-end multiresolution dataflow strategy whose goal is to increase efficiencies in practice by several orders of magnitude. Given recent advancements in subdivision-surface wavelet compression and view-dependent optimization, we present algorithms here that provide the "glue" that makes this strategy hold together. Shrink-wrapping converts highly detailed unstructured surfaces of arbitrary topology to the semi-structured form needed for wavelet compression. Remapping to triangle bintrees minimizes disturbing "pops" during realtime display-triangulation optimization and provides effective selective-transmission compression for out-of-core and remote access to these huge surfaces.

1 Background

Terascale physics simulations are now producing tens of terabytes of output for a several-day run on the largest computer systems. An example: the Gordon Bell Prize-winning simulation of a Richtmyer-Meshkov instability in a shock-tube experiment [5] produced isosurfaces of the mixing interface with 460 million unstructured triangles using conventional extraction methods. New parallel systems are three times as large as when this run took place, so billion-triangle surfaces are to be expected shortly. Since we are interested in interaction, especially sliding through time, surfaces are precomputed (if they were not, the 100 kilotriangle-per-processor rates for the fastest isosurface extractors would result in several minutes per surface on 25 processors). Using 32-bit values for coordinates, normals and indices requires 16 gigabytes for a single surface, and several terabytes for a single surface tracking through all 274 time steps of the simulation. This already exceeds the compressed storage of the 3D fields from which the surfaces are derived, and adding additional isolevels or fields per time step would make this approach infeasible. With the gigabyte-per-second read rates of current RAID storage, it would take 16 seconds to read a single surface. A factor of 100 compression with no perceptible loss would cleanly solve both the storage and load-rate issues. This may be possible with new bicubic subdivision-surface wavelets [1].

Another bottleneck occurs with high-performance graphics hardware. The fastest commercial systems as of this writing can effectively draw around 20 million triangles per second, i.e. around 1 million triangles per frame at 20 frames per second. Thus almost a thousand-fold reduction in triangle count is needed. This level of reduction is too aggressive to be done without taking the interactively-changing viewpoint into account. As the scientist moves close to a feature of interest, that feature should immediately and seamlessly be fully resolved while staying within the interactive triangle budget. This can be formulated as the optimization of an adaptive triangulation of the surface to minimize a view-dependent error measure such as the projected geometric distortion on the screen. Since the viewpoint changes 20 times per second, and the error measure changes with the viewpoint, the million-element adaptation must be re-optimized continuously and quickly. The fastest theoretic time

[1]Center for Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551, USA, duchaineau1@llnl.gov

[2]AG Computergraphik und Computergeometrie, Universitt Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany, bertram@informatik.uni-kl.de

[3]Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California at Davis, Davis, CA 95616-8562, USA, {hamann,joy}@cs.ucdavis.edu

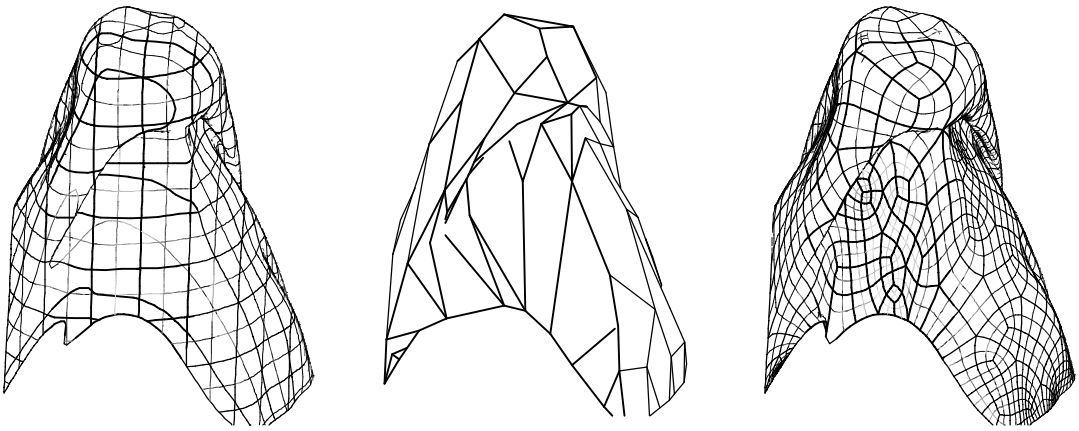


Figure 2: Shrink wrapping steps (left to right): the full-resolution isosurface, the base mesh constructed from edge collapses, and the final shrink-wrap with subdivision-surface connectivity.

that any algorithm can perform this optimization is $O(\Delta \text{output})$, an amount of time proportional to the number of element changes in the adaptation per frame. The ROAM algorithm achieves this optimal time using adaptations built from triangle bintrees [3]. With this approach in a flight-simulation example, around 3% of the elements change per frame, resulting in a thirty-times speedup in optimization rates. This is critically important since the bottleneck in fine-grained optimizers is processing time rather than graphics hardware rates.

The wavelet compression and view-dependent optimization are two powerful tools that are part of the larger dataflow from 3D simulation to interactive rendering. These are by no means the only challenging components of a terascale visualization system. For example, conversion is required to turn irregular extracted surfaces into a form appropriate for further processing (see Figure 1). Before turning to our focus on the two surface-remapping steps that tie together the overall dataflow, we give an overview of the complete data pipeline for surface interaction.

2 End-to-End Multiresolution Dataflow

The processing steps required to go from massive 3D field data to the interactively-changing optimal triangulations sent to graphics hardware involve six steps:

Extract: get unstructured triangles through accelerated isosurface extraction methods or through material-boundary extraction from volume-fraction data [2].

Shrink-wrap: convert the high-resolution unstructured triangulation to a similarly detailed surface that has subdivision-surface connectivity (i.e. is *semi-regular*), has high parametric quality, and minimizes the number of structured blocks. This uses three phases: (1) compute the signed-distance transform of the surface, (2) simplify the surface to a base mesh, and (3) iteratively subdivide, smooth and snap the new mesh to the fine-resolution surface until a specified tolerance is met.

Wavelet compress: for texture storage, geometry archiving, and initial transmission from the massively-parallel code runs, use high-order wavelets based on subdivision surfaces for nearly-lossless compression.

Triangle bintree re-map: re-map the shink-wrap parameterization to be optimal for subsequent view-dependent optimization (this is different than being optimal for high-quality wavelet compression). This involves piecewise-linear "wavelets" without any vanishing moments, but where most wavelet coefficients can be a single scalar value in a derived normal direction, and where highly localized changes are supported during selective refinement.

Selective Decompress: asynchronously feed a trickle of compressed detail where the view-dependent adaptation is most likely to find missing values during selective refinement in the near future. This trickle can be efficiently stored in chunks for efficient I/O or network transmission.

Display-list optimization: perform the realtime optimization of the display-list adaptive triangulation each frame, making maximal use of frame-to-frame coherence to accelerate frustum culling, element priority computation, and local refinement and coarsening to achieve the optimum per-view geometry.

The six processing steps occur in the order listed for terascale surface visualization. The first three steps—extraction, shrink-wrap and wavelet compress—typically occur in batch mode either as a co-process of the massively parallel physics simulation, or in a subsequent parallel post-processing phase. Given the selective access during interaction, and given the already existing wavelet hierarchy, the remapping for ROAM interaction can happen on demand given modest parallel resources. Because the ROAM remapping works in time $O(\text{output})$ in a coarse-to-fine progression, the amount of computation per minute of interaction is independent of the size of the physics grid or the fine-resolution surfaces. The ROAM remapper is envisioned as a runtime data service residing on a small farm of processors and disks, that reads, remaps and feeds a highly compact data stream on demand to the client ROAM display-list optimizer. This server-to-client link could be across the campus LAN or over high-speed WAN to provide remote access. the ROAM algorithm works at high speed per frame to optimize the display triangulation, and thus will reside proximate to the graphics hardware.

The remainder of this paper focuses on the two remapping steps in this dataflow. The first prepares a surface for wavelet compression, the second for ROAM triangle bintree optimization.

3 Shrink-Wrapping Large Isosurfaces

Before subdivision-surface wavelet compression can be applied to an isosurface, it must be re-mapped to a mesh with subdivision-surface connectivity. Minimizing the number of base-mesh elements increases the number of levels in the wavelet transform, thus

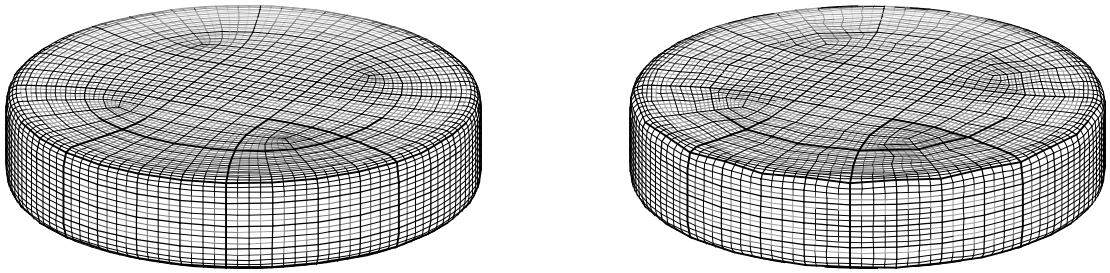


Figure 3: Before (left) and after the remapping for triangle bintree hierarchies. Tangential motion during subdivision is eliminated.

increasing the potential for compression. Because of the extreme size of the surfaces encountered, and the large number of them, the re-mapper must be fast, work in parallel, and be entirely automated. The compression using wavelets is improved by generating high-quality meshes during the re-map. For this, highly smooth and non-skewed parameterizations result in the smallest wavelet coefficient magnitudes, yielding small outputs after entropy coding. In addition, we would like to allow the new mesh to optionally have simplified topology, akin to actual physical shrink-wrapping of complex 3D objects with a few connected sheets of plastic.

The algorithm we propose for this shrink-wrapping is an elaboration of the method described in [1]. That method was used to demonstrate wavelet transforms of complex isosurfaces in a non-parallel, topology-preserving setting. The algorithm takes as input a scalar field on a 3D mesh and an isolevel, and provides a surface mesh with subdivision-surface connectivity as output, i.e. a collection of logically-square patches of size $(2^n + 1) \times (2^n + 1)$ connected on mutual edges. The algorithm at the high level is organized in three steps:

Signed distance transform: for each grid point in the 3D mesh, compute the signed-distance field, i.e. the distance to the closest surface point, negated if in the region of scalar field less than the isolevel. Starting with the vertices of the 3D mesh elements containing isosurface, the transform is computed using a kind of breadth-first propagation. Data parallelism is readily achieved by queueing up the propagating values on the boundary of a subdomain, and communicating to the block-face neighbor when no further local propagation is possible.

Determine base mesh: To preserve topology, edge-collapse simplification is used on the full-resolution isosurface extracted from the distance field using conventional techniques. This is followed by an edge-removal phase (edges but not vertices are deleted) that improves the vertex and face degrees to be as close to four as possible. Parallelism for this mode of simplification is problematic, since edge-collapse methods are inherently serial using a single priority queue to order the collapses.

To allow topology reduction, the 3D distance field is simplified before the isosurface is extracted. Simplification can be performed by using wavelet low-pass filtering on regular grids, or after resampling to a regular grid for curvilinear or unstructured 3D meshes. The use of the signed-distance field improves the simplified-surface quality compared to working directly from the original scalar field. To achieve the analog of physical shrink-wrapping, a max operation can be used in place of the wavelet filtering. This form of simplification is easily parallelized in a distributed setting.

Subdivide and fit: The base mesh is iteratively fit and the parameterization optimized by repeating three phases: (1) subdivide using Catmull-Clark rules, (2) perform edge-length-weighted Laplacian smoothing, and (3) snap the mesh vertices onto the original full-resolution surface with the help of the signed-distance field. Snapping involves a hunt for the nearest fine-resolution surface position that lies on a line passing through the mesh point in an estimated normal direction of the shrink-wrap mesh. The estimated normal is used, instead of e.g. the distance-field gradient, to help spread the shrink-wrap vertices evenly over high-curvature regions. The signed-distance field is used to provide Newton-Raphson-iteration convergence when the snap hunt is close to the original surface, and to eliminate nearest-position candidates whose gradients are not facing in the directional hemisphere centered on the estimated normal. Steps 2-3 may be repeated several times after each subdivision step to improve the quality of the parameterization and fit. In the case of topology simplification, portions of surface with no appropriate snap target are left at their minimal-energy position determined by the smoothing and the boundary conditions of those points that do snap. Distributed computation is straightforward since all operations are local and independent for a given level of resolution.

The shrink-wrap process is depicted in Figure 2 for approximately .016% of the 460 million-triangle Richtmyer-Meshkov mixing interface in topology-preserving mode. The original isosurface fragment contains 37,335 vertices, the base mesh 93 vertices, and the shrink-wrap result 75,777 vertices.

4 Re-Mapping for ROAM

The Realtime Optimally Adapting Meshes (ROAM) algorithm typically exploits a piecewise block-structured surface grid to provide efficient selective refinement for view-dependent optimization. A triangle bintree structure is used. This consists of a hierarchy of logically right-iscoceles triangles, paired across common base edges at a uniform level of subdivision. A simple split operation bisects the common base edge of such a pair, turning the two right-isosceles triangles into four. Merging reverses this operation. This is depicted in Figure 3.

The shrink-wrapping process that we have described produces meshes that are technically in this form, but cause large tangential motions of the mapping during refinement even in regions of flat geometry. To correct for this, we have devised a new remapping algorithm that eliminates tangential motion altogether whenever possible during ROAM refinement, but never causes the mapping to become degenerate or ill-defined. In effect, the surface is defined by a series of neighborhood height (normal) maps, allowing details to be stored with a single scalar rather than a 3-component displacement vector. Our method is similar to the independent work of Guskov *et al.* [4], differing primarily in the driving goal (efficient view-dependent optimization with crude wavelet compression in our case) and the details of mesh structure, subdivision scheme supported, intersection acceleration an so on.

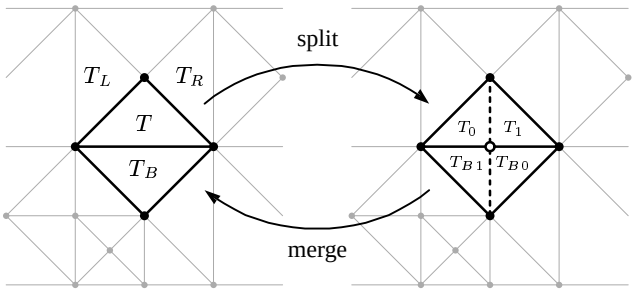


Figure 4: Split and merge operations on a bintree triangulation. A typical neighborhood is shown for triangle T on the left.

The normal-remapping works from coarse to fine resolutions, remapping a complete uniform level of the hierarchy at once. The vertices of the base mesh are left fixed at their original positions. For every edge-bisection vertex formed in one subdivision step, estimated normals are computed by averaging the normals to the two triangles whose common edge is being bisected. For every vertex that has been remapped, its patch and parameter coordinates in the original map are kept. During edge bisection, the *parametric midpoint* is computed by topologically gluing at most two patches together from the original mesh, computing the mid-parameter values in this glued space, then converting those parameters back to unglued parameters. Given the constraints on our procedure, it is not possible for bisecting-edge endpoints to cross more than one patch boundary. A ray-trace intersection is performed from the midpoint of the line segment being bisected, in the estimated normal direction. Since we expect the intersection to be near the parametric midpoint in most cases, it is efficient to begin the ray intersection tests there for early candidates. Since the surface being ray-traced stays fixed throughout the remapping, the construction of typical ray-surface intersection-acceleration structures can be amortized and overall offer time savings (reducing the time from $\mathrm{O}(N \log(N))$ to $\mathrm{O}(N)$ for N mesh vertices). Interval-Newton and finally Newton-Raphson iterations can be performed for the final precise intersection evaluation. Intersections are rejected if they are not within a parametric window defined by the four remapped vertices of the two triangles being split, shrunk by some factor (e.g. $.5$) around the parametric midpoint. The closest acceptable intersection is chosen. If none exist or are acceptable, the parametric midpoint is chosen.

The result of remapping is shown in Figure 4 for a test object produced by Catmull-Clark subdivision with semi-sharp features. The original parameterization on the left is optimal for compression by bicubic subdivision-surface wavelets, but produces extreme and unnecessary tangential motions during triangle-bintree refinement. The remapped surface, shown on the right, has bisection-midpoint displacements ("poor man's wavelets") of length zero in the flat regions of the disk, and displacements of minimized length elsewhere. We note that while the main motivation for this procedure is increasing accuracy and reducing the "pops" during realtime display-mesh optimization, the typical reduction to a single scalar value of the displacement vectors (wavelet coefficients) gives a fair amount of compression. This is desirable when the re-map from high-quality wavelet parameterization and compression is too time-consuming such as on the client end of the server-client asynchronous dataflow described earlier.

We note that the ROAM algorithm naturally requires only a tiny fraction of the current optimal display mesh to be updated each frame. Combined with caching and the compression potential of the remapping, this promises to provide an effective mechanism for out-of-core and remote access to the surfaces on demand during interaction.

5 Future Work

Several pieces of the terascale dataflow strategy have been realized to date, but many challenges remain to create a full capability:

1. For topology-preserving simplification, the inherently serial nature of the queue-based schemes must be overcome to harness parallelism.

2. Transparent textures or other means must be devised to handle the un-mapped surface regions resulting from topology-simplifying shrink wrapping.

3. The shrink-wrapping procedure can fail to produce one-to-one, onto mappings in some cases even when such mappings exist. Perhaps it is possible to revert to expensive simplification schemes that carry one-to-one, onto mappings only in problematic neighborhoods.

4. Shrink-wrapping needs to be extended to produce time-coherent mappings for time-dependent surfaces. This is a great challenge because of the complex evolution that surfaces go through during physics simulations.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. We thank LLNL for support through the Science and Technology Education Program, the Student-Employee Graduate Research Fellowship Program, the Laboratory-Directed Research and Development Program, the Institute for Scientific Computing Research, and the Accelerated Strategic Computing Initiative.

References

[1] Martin Bertram, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. *Proceedings of IEEE Vis00*, October 2000.

[2] Kathleen S. Bonnell, Daniel R. Schikore, Kenneth I. Joy, Mark Duchaineau, and Bernd Hamann. Constructing material interfaces from data sets with volume-fraction information. *Proceedings of IEEE Vis00*, October 2000.

[3] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. *IEEE Visualization '97*, pages 81–88, November 1997. ISBN 0-58113-011-2.

[4] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, July 2000.

[5] Arthur A. Mirin, Ron H. Cohen, Bruce C. Curtis, William P. Dannevik, Andris M. Dimits, Mark A. Duchaineau, Don E. Eliason, Daniel R. Schikore, Sarah E. Anderson, David H. Porter, Paul R. Woodward, L. J. Shieh, and Steve W. White. Very high resolution simulation of compressible turbulence on the IBM-SP system. *Supercomputing 99 Conference*, November 1999.

Bjoern Heckel

Epicentric, Inc.
333 Bryant Street, Suite 300
San Francisco, CA 94107

Abstract

Clustering-based Multiresolution Methods for Scientific Visualization

In this presentation, a generic framework for the creation of hierarchical data representations for applications in scientific visualization is described. The framework is based on clustering, which is a fundamental technique in classical data analysis and statistics. The major strength of the clustering-based approach for the construction of hierarchical representations for very large scientific data sets is its universal applicability. By using different error metrics, in combination with altering parameters of the clustering process, the clustering method can be used to solve a wide range of problems in scientific visualization and hierarchical data exploration. Adaptations of the general clustering framework were developed and used for the specific purposes of (i) feature extraction from volumetric data sets; (ii) surface reconstruction from digitized/scanned objects; (iii) simplification/coarsening of surface meshes; (iv) simplification of vector field data in three-dimensional space; and (v) simplification of scalar field data in three-dimensional space.

The clustering framework supports the construction of data hierarchies for all commonly encountered discrete data formats used throughout scientific and engineering applications. This includes data sets given simply as a set of points without any connectivity information (scattered data) as well as data defined on classical finite element/finite difference meshes (rectilinear meshes, curvilinear meshes, unstructured meshes, triangular and tetrahedral meshes, etc.) The general applicability of the approach makes it superior to most other existing methods used today for the construction of hierarchical data representations.

The output of the clustering process, the *cluster tree*, can be used for view-dependent and adaptive rendering. View-dependent and adaptive rendering is concerned with the utilization of different data resolution levels in different regions of a data set to be visualized. As a consequence, one has to access nodes from different levels of the cluster tree to enable rendering at different resolutions in different regions in space. By coupling the view-dependent/adaptive paradigm with a scalable implementation of the hierarchy-building process, it is possible to create and interactively explore very large scientific data hierarchies in greatly reduced response times.

The concept of "visualization of scientific data without meshes" is introduced. Traditionally, different visualization algorithms were developed to create visualizations of data defined on various kinds of underlying meshes. The dependency of visualization algorithms on the underlying mesh structure (connectivity among points in space) has produced a multitude of mesh-specific rendering methods. The need to develop a framework supporting the hierarchical visualization of scientific data without dependency on point connectivity existed for a long time. The solution is proposed in this talk: instead of developing visualization algorithms for specific mesh structures, we base the visualization process entirely on the concept of merely rendering scattered data, i.e., we ignore possibly existing original point connectivity. Based on this paradigm, we have created a new approach for both the construction and utilization of scientific data hierarchies. We do not consider/establish point connectivity during hierarchy generation, nor do we require a mesh to be known prior to invoking an interactive data exploration process accessing a generated "meshless" data hierarchy.

We demonstrate the value of meshless hierarchical visualization for three-dimensional scalar and vector field data sets. The scientific data visualization process typically requires local analytical field approximation, which we can compute highly efficiently by constructing localized scattered data approximations, which are analytical field descriptions obtained from discrete point date without connectivity information. The meshless paradigm for hierarchy construction and storage is extremely space-efficient.

We introduce the concept of *visualization-space error metrics*. Traditional numerical analysis and approximation theory base the quality of an approximation of a function by considering the deviation of the approximation and the function being approximated in the space/domain over which they are defined. In the context of data approximation having the purposes of visualization in mind, another measure for "approximation quality" might be appropriate: an approximation of some original discrete scientific data set is a good approximation when the visualizations of the approximation and the original data set differ minimally. By considering, during the stage of hierarchy construction, the visualization methods that will eventually be used for rendering, it is possible to construct a visualization-method-dependent data hierarchy — whose visualization will allow the best possible visualization at each level of resolution.

Mesh Edge Detection

Andreas Hubeli, Kuno Meyer, Markus Gross
Department of Computer Science, ETH Zurich, Switzerland

Abstract

We present a framework to extract line-type features from unstructured two-manifold meshes. Our method computes a collection of piecewise linear curves describing the salient features of the mesh, such as edges and ridge lines. Our algorithms are semi-automatic, that is, they require the user to input a few control parameters and to select the operators to be applied to the mesh.

Our mesh edge detection algorithm can be used as a preprocessor for a variety of applications including mesh fairing and smoothing.

1 Introduction

Recent advances in acquisition systems have resulted in the ready creation of very large, densely sampled surfaces, usually represented as triangle meshes. The impossibility of real-time interaction with these large models has motivated many researchers in the computer graphics community to design advanced mesh processing methods including subsampling, restructuring, fairing and others. The early approaches, such as the *vertex removal* algorithm of W. Schröder [7] or the *progressive mesh* algorithm of H. Hoppe [4], use local error norms to construct multiresolution approximations of meshes by iteratively removing information from the input mesh.

More recent representations are based on the generalization of fairing techniques from signal processing [8], [6], [5]. They use multiresolution algorithms to improve the mesh approximation. In this paper we investigate a related problem: *mesh edge detection*. Our goal is to extract line-type features from meshes which can be used to construct more sophisticated multiresolution representations. The most important advantage of using line-type features is that we can force fairing algorithms to retain feature information, such as sharp edges or ridge lines - much in the same way it was accomplished for subdivision surfaces. We achieve this goal by generalizing well known computer vision techniques, such as [1], to meshes with arbitrary connectivity.

The paper is organized as follows: in section 2 we present the first major component of our framework, the set of classification operators. In section 3 we introduce the second component of our framework, the detection operators. In section 4 we describe some of the experimental results we obtained using this technique. Finally, we discuss some work in progress.

E-mail: hubeli@inf.ethz.ch
meyerk@student.ethz.ch
grossm@inf.ethz.ch

Address: Department of Computer Science
ETH Zentrum
CH - 8092 Zurich

2 Classification Phase

The classification operators basically assign a *weight* to every edge in the mesh. The weight is proportional to the importance of the edge: ideally, edges close to or on line-type features should be assigned large weights whereas all remaining edges should get small weights. The weights will then be used to extract the subset of the "most important" edges, the so-called *feature edges*. They are employed in the detection phase to construct the line-type mesh features.

In the following subsections we will describe some of the classification operators we designed and tested.

2.1 Second Order Difference (SOD)

This most simple operator assigns a weight to every edge in the mesh proportional to the dihedral angle defined by the normals of its two adjacent triangles. The idea is similar to the second order difference operator constructed by Guskov et al. in [2] which was used to fair meshes of arbitrary connectivity. The operator, described by equation (1), is locally bound and can be evaluated efficiently.

$$w(e) = \cos\left(\frac{\boldsymbol{n}_i}{\|\boldsymbol{n}_i\|} \cdot \frac{\boldsymbol{n}_j}{\|\boldsymbol{n}_j\|}\right)^{-1} \tag{1}$$

$\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ correspond to the normals of the two triangles that share edge e.

This technique is well suited for coarse, pre-optimized meshes. SOD however, performs poorly on very smooth or noisy meshes, since all computations are carried out within a small region of support.

The results in figure 5.a were generated using this operator.

2.2 Extended Second Order Difference (ESOD)

With only a little effort we can build a simple extension to the previous operator. Instead of using the normal of the two neighboring triangles, we take the average normals computed from the one-ring of the vertices opposite to the edge and apply them to equation (1). This is shown in figure 1.

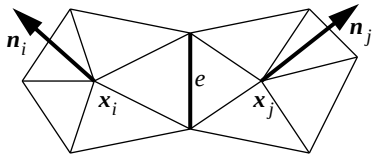


Figure 1: Support of the extended second order difference operator

The increase in the size of the support has the expected impact: the influence of noise on the classification process is significantly attenuated. However, as a drawback, ESOD does not perform that well on coarse meshes.

2.3 Best Fit Polynomial (BFP)

In this approach all the vertices used to evaluate the classification operator on an edge are projected onto a two-dimensional parameter plane. The projected vertices are then interpolated with a best fit polynomial $p(u)$ of degree n. Finally the curvature of the (planar) polynomial is evaluated at the edge position e, as described by equation (2):

$$w(e) = p''(e) \tag{2}$$

The major difficulties of this approach are the definition of the parameter plane and the proper projection of the initial vertices from 3-space. An intuitive definition of the parameter space is given in figure 2:

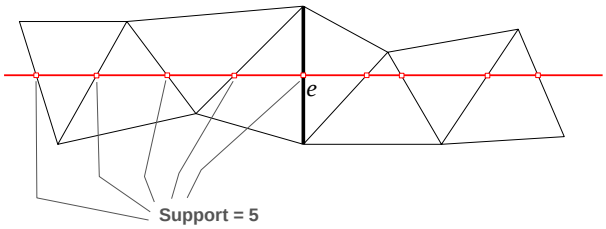


Figure 2: Illustration of the parameter plane used for the BFP method (top view)

We propose to set the parameter plane to be perpendicular to the edge being considered. In addition, the midpoint of the edge is defined to lie on the plane. The points used in the best fit process are computed from the intersection of the plane with a set of neighboring triangle edges. The most important advantage of this strategy is that the support of the operator can be chosen freely. That is, we can even adapt it locally for each edge. Furthermore, the degree of the fitting polynomial can be adjusted to the size of the support.

Both figure 5.b and figure 5.c were generated with variations of this operator.

2.4 Angle Between Best Fit Polynomials (ABBFP)

The last operator is an extension to the previously introduced polynomial fit. As in the previous case, polynomials are fitted through the parameter plane of every edge. This operator actually fits two polynomials: one for the vertices that lie on one side of the edge, and one for the vertices lying on the other. The weight assigned to the edge is then chosen to be proportional to the angle between the two curve tangents evaluated at the edge position. It yields according to equation (3):

$$w(e) = \cos\left(\frac{(1, p_l'(e))}{\|(1, p_l'(e))\|} \cdot \frac{(1, p_r'(e))}{\|(1, p_r'(e))\|}\right)^{-1} \tag{3}$$

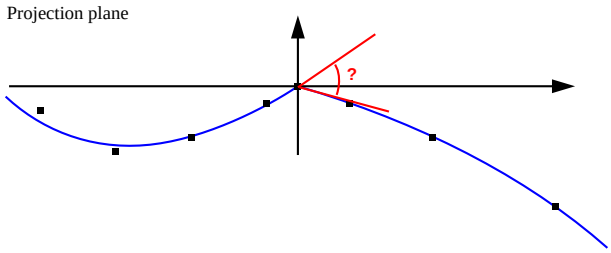


Figure 3: ABBFP operator

All the results in figure 6 were generated using this operator and with support sizes ranging from 0.25% (figure 6.a) to 4% (figure 6.c) of the size of the object bounding box.

3 Detection Phase

As previously discussed, the classification operators are used to assign a weight to every edge in the mesh. The larger the weight, the "more important" the edge. In a second phase, proper feature lines need to be constructed. This is accomplished in three steps:

- First, a subset of *feature edges* is constructed. Only edges in this set will be further considered to compute the final set of line-type features.
- Next, the feature edges are clustered into *patches*. These patches define mesh regions where line-type features are present.
- Finally, the *line-type features* are extracted using a skeletonizing algorithm that reduces the patches to piecewise linear curves.

3.1 Selection of Feature Edges

This process is heavily mesh and user dependent, since the number of feature edge candidates depends both on the size and geometry of the mesh and on the user's intention. Hence, we require the user to select appropriate threshold values. We propose the following two different strategies for thresholding:

Standard Thresholding

The most simple thresholding approach analyzes every edge separately. Based on a user-provided parameter it decides whether an edge is a feature edge. The user can specify the threshold parameter both in percentage of edges that must be preserved, or as the minimal weight that an edge must have to be included into the subset of feature edges.

Hysteresis Thresholding

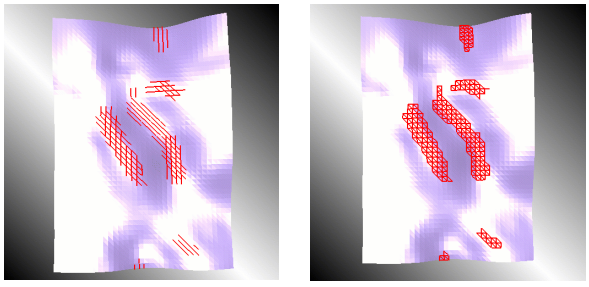
This type of approach uses two threshold values serving as a lower and an upper bound of a hysteresis. If the weight associated with an edge is larger than the upper value, the edge is automatically defined as a feature edge. If the weight is smaller than the lower bound, then the edge is automatically discarded. The remaining edges, whose weights lie between the lower and upper bounds, are only defined as feature edges if one or more of their neighbors are feature edges. The advantage of this approach is that it constructs smoother patches in regions where feature edges are present.

3.2 Construction of the Patches

The patches are generated from the subset of feature edges using the following approach:

- every edge that shares *both* of its vertices with feature edges is inserted into the subset of feature edges.

As an example consider the figure 4. The results illustrated in figure 4.b demonstrate that the patches are filled with feature edges and that the size of the patches is clearly bound.



a) b)

Figure 4: Construction of the patches:
a) Subset of feature edges computed by the hysteresis thresholding scheme
b) Set of patches generated using the method of section 3.2

3.3 Skeletonizing

As described the output of the previous step is a set of patches. As such, a patch is a collection of edge features describing the neighborhood of one or more line-type features. The final step of the detection phase is to reduce these patches to a set of line-type features. We accomplished this using the following two skeletonizing strategies:

Triangle Based Skeletonizing

This approach reduces patches to lines of a thickness smaller or equal to 2 by removing triangles from the patch - one at a time. In order to get the correct result, we have to impose constraints on the removal strategy. The drawback of this approach is that it might break up larger line-type features into multiple disconnected segments. The advantage is that can be computed very efficiently. A more detailed description has to be omitted for brevity.

Vertex Based Skeletonizing

In the second approach, we use potential fields to remove vertices from the patch. In a first step, a potential field is defined for every vertex whose strength is proportional to its distance from the patch boundary. Next, we discard all the vertices that have at least one neighbor with a larger potential. As a result we obtain a surviving subset of disconnected vertices. These vertices are subsequently glued together using certain assumptions on the potential fields. Again, further details have to be omitted.

4 Results

In this section we will present and discuss experimental results obtained on different meshes. We used both well known meshes, such as the bunny, mannequin, motor part, and the golf club, as well as a simple, smooth geological surface. Most of the meshes have an arbitrary connectivity, and they exhibit some sharp line-type features. The geological surface is the only exception, being a regular grid and not having any salient features.

In the first sequence of pictures presented in figure 5 we compare some of the operators defined in section 2. On this mesh we obtained the best results by using operators with small support. This is due to the fact that the model is only sparsely sampled. As a consequence, operators with large support include information that is geometrically too far away from the edge. This problem was addressed by constraining the support of the BFD operators. Both the SOD operator and the BFP operators generated good results.

In the second sequence illustrated in figure 6 we capture the influence of the support of the ABBFP operator. From left to right we show the influence of increasing the support from 0.25% to 4% of the size of the bounding box. In figure 6.a the support is too small, and as a consequence, the noise of the mesh corrupts the performance of the classification operator. At the other extreme, in figure 6.c the support of the operator is too large. Hence, the marked edges are clustered around only a few of the prominent mesh features. The best results have been achieved by setting the support to an intermediate value of 2%, as shown in figure 6.b.

Finally, figure 7 depicts the results obtained on different meshes by combining the classification and detection operators appropriately. Interestingly, the line-type features of the head mesh of figure 7.a include the outlines of the eyes, mouth, ears, nose and eyebrows - as we would expect it from a feature extraction algorithm. The features of the golf club mesh of figure 7.b have been recognized correctly, and all the sharp edges have been captured. Our last image in figure 7.c shows a geological surface. Although the surface does not contain any salient edges, our framework extracted a few features. By comparing these results to the actual structure of the surface, we found that the algorithm extracted local ridges and valley lines in the mesh.

5 Conclusions and Future Work

In this paper we presented a framework for the detection of line-type features in meshes of arbitrary connectivity. We proposed a two-stage process consisting of a *classification phase* and a *detection phase*. In order to handle a variety of different meshes, we provide a set of operators with different properties.

As already mentioned, the user must select the operators as well as a few parameters for the classification and detection steps. As such, the process is not fully automatic. It still requires manual assistance and tuning for optimal performance. We do not consider this as a major drawback, since all algorithms can be computed efficiently.

The presented results are encouraging, and we expect some major improvements in the near future. The most important ones include:

- Computational efficiency: although the operators are fairly efficient, it still takes a few seconds (up to a few minutes for very large meshes) to compute the features. This is certainly a drawback for interactive applications where a user might try different operators to optimize the results.
  We will tackle this problem by designing a multiresolution representation of the mesh. Since the multiresolution representation effectively strives towards maintaining model features, we can identify efficiently which edges need to be considered for the computation.
- Improved classification strategy: the most important problem encountered in the classification phase is that features are not necessarily high frequency information. This is well known from image edge detection. Although, intuitively, we tried to capture low-frequency information by extending the support of the operators improvements might be obtained by mesh decomposition [3].
  The advantage of such settings is that one could start the classification process on a coarse, low-frequency approximation and progressively improve the results while refining the mesh.
- For computational efficiency, our current skeletonizing strategies are based on topological distance. Taking geometric distances instead might further improve the performance of the framework.
- Our final goal is, of course, to provide a mesh analysis sophisticated enough to automatically determine the optimal operators and parameters.

Literature

[1] J. Canny. “A computational approach to edge detection.” In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 6, pages 679-696, Nov. 1986.

[2] I. Guskov, W. Sweldens, and P. Schröder. “Multiresolution signal processing for meshes.” In *SIGGRAPH ’99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[3] M. Gross and A. Hubeli. “Eigenmeshes.” Technical Report ETH Zurich, Mar. 2000.

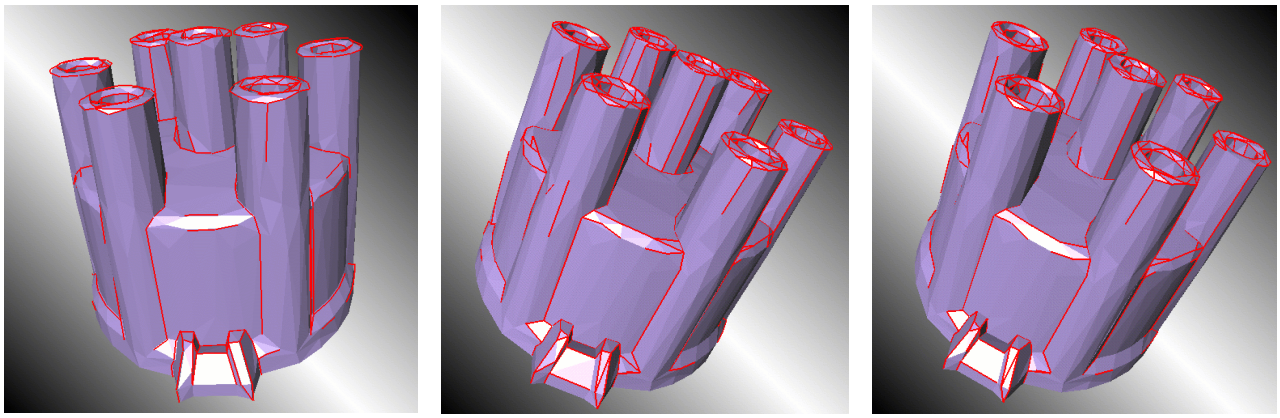
[4] H. Hoppe. “Progressive meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[5] A. Hubeli and M. Gross. “Fairing of non-manifolds for visualization.” In Visualization 2000 Proceedings, Oct 2000.

[6] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. “Interactive multi-resolution modeling on arbitrary meshes.” In M. F. Cohen, editor, *SIGGRAPH 98 Conference proceedings*, Annual Conference Series, pages 105–114. ACM Press and Addison Wesley, July 1998.

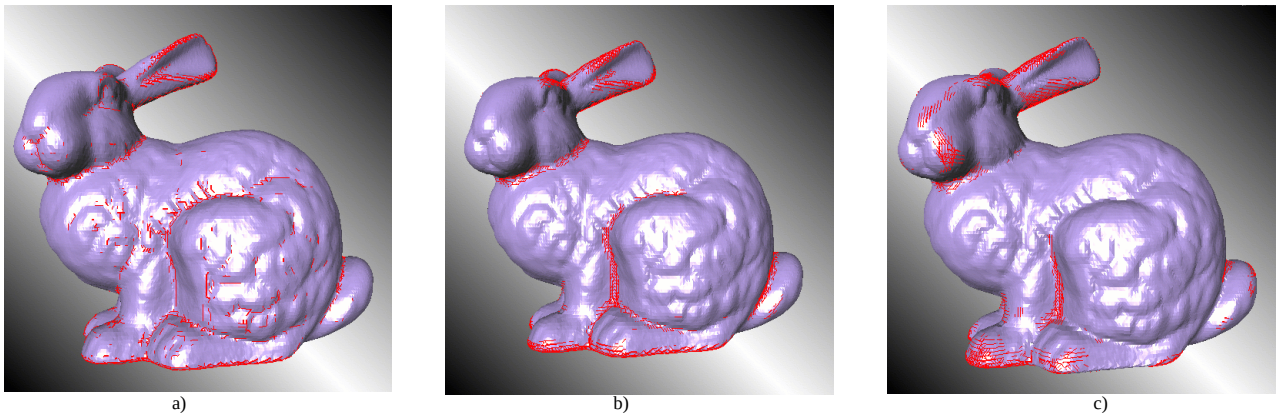
[7] W. Schröder, J. Zarge, and W. Lorensen. “Decimation of triangle meshes.” In SIGGRAPH 92 Conference Proceedings, Annual Conference Series, pages 65–70, July 1992.

[8] G. Taubin. “A signal processing approach to fair surface design.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995.



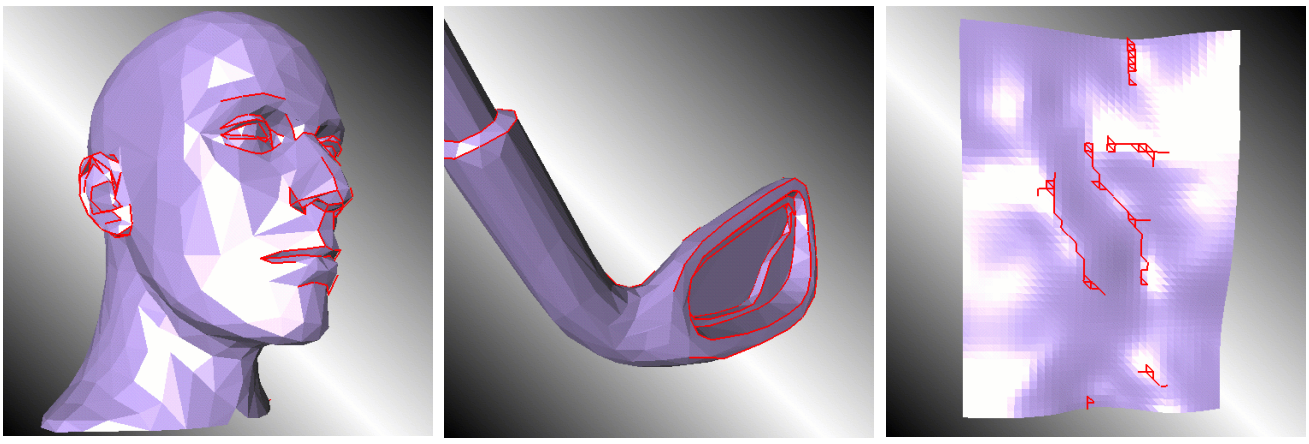
a) b) c)

Figure 5: Results computed on a motor part using different operators:
a) Line-type features computed using the SOD operator
b) Line-type features computed using an *extension to the BFP* operator
c) Line-type features computed using an *extension to the BFP* operator



a) b) c)

Figure 6: Results computed on the bunny model using the *ABBFP* operator:
a) Edges detected using a support of 0.25% of the size of the object bounding box
b) Edges detected using a support of 1.0% of the size of the object bounding box
c) Edges detected using a support of 4.0% of the size of the object bounding box



a) b) c)

Figure 7: Mesh edge detection applied to other models:
a) A mannequin head. Feature lines detected: eyes, nose, lips, ears, eyebrows
b) A golf club. All the sharp features have been detected correctly
c) A geological surface. Some interesting feature-lines could be extracted from this very smooth surface

Critical Points at Infinity: a missing link in vector field topology

Issac Trotts, David Kenwright, Robert Haimes

Massachusetts Institute of Technology

This paper presents an improved technique for extracting singular streamlines that extends the capabilities of vector field topology. Singular streamlines are a unique set of tangent curves that connect singular or critical points. In prior work [Ken99], we demonstrated that vector field topology (a well-known technique for extracting singular streamlines) failed to identify a class of singular streamlines because they did not originate from critical points inside the computational domain. In this paper, we show that singular streamlines that originate from critical points at infinity are the missing link and account for the previously undetectable "open" singular streamlines.

1. Background

Automated feature extraction is an emerging field of research in which meaningful features or structures are automatically extracted from large-scale computational simulations. The aim of feature extraction is to develop data analysis tools that automatically deduce the location, shape, and strength of specific features without human intervention, and do so in substantially less time than their human counterpart. Feature extraction algorithms are programmed with domain-specific knowledge, so they do not require a-priori knowledge of places to look for interesting behavior. By extracting only the salient features from a vector field, computational datasets can be distilled by a factor of 10^5 [Ken99]. The level of data reduction can be increased beyond 10^6 by fitting curves to the raw geometry that is produced by feature extraction algorithms. By employing feature extraction as either a co-processing [Hai95] or post-processing [Ken99] data analysis tool, it is possible to interactively visualize a terascale data set using inexpensive desktop computers.

The long-term objective of this research is to develop feature extraction tools to construct the entire vector field topology, consisting of singular points, lines, and surfaces, for 3-D and 4-D computational simulations. We are primarily interested in large-scale vector fields in computational fluid dynamics (CFD) where these features are commonly called critical points, separation lines, and separation surfaces respectively. The same geometric features can be found in other scientific fields such as electromagnetics, geomorphology, and quantum chemistry, although they given different names [Rot00].

Singular stream-lines and -surfaces are immensely useful geometric features because they partition vector fields into topologically-like regions in which the streamlines and stream surfaces have a qualitatively similar behavior. They help scientists and engineers understand the mechanisms of flow separation, vortex genesis, and the formation of turbulence. Scientists often resort to hand drawn pictures to explain these phenomena because of the lack of automated tools. At present, there are no robust computational methods to extract a complete set of singular streamlines and stream surfaces from numerical simulations of fluid flows.

One of the early and most prominent feature extraction techniques is based on the classification of critical points. A critical point is a local feature that occurs where the velocity vector field is zero. In 2-D, the topology of a vector field, $\mathbf{u}$, consists of critical points and the tangent curves (instantaneous streamlines) that connect these points. Critical points are in fact degenerate streamlines, and therefore the only points in vector fields where other streamlines may start or end, except for the boundary. Because the velocity at a critical point is zero, the velocity field in the neighborhood of the critical point is determined by the Jacobian tensor $\nabla\mathbf{u}$. Critical points are classified, to a first-order approximation, by the eigenvalues and eigenvectors of $\nabla\mathbf{u}$.

The two-dimensional topology of a vector field is constructed by finding the set of streamlines that start or end at critical points. Helman and Hesselink [Hel91] and Globus *et al.* [Glo91] showed that (closed) separation lines could be generated by

integrating outwards from the "saddle" critical points in the real eigenvector directions. These tangent curves were classified as either separation or re-attachment lines based on the sign of the eigenvalues: positive (repelling) for attachment, and negative (attracting) for separation. The vector field topology conveys the essential behavior of a 2-D flow field by leaving out some geometric details about the flow in these regions.

The concept behind vector field topology is to divide the flow into closed regions in which each of the streamlines is "similar" to its neighbor. Closed separation lines are defined as singular streamlines that start and end at singular points. Experimental results from wind- and water tunnels revealed another type of singular streamline that did not start or end at a singular point. These "open" separation lines were not detectable by vector field topology methods that originated from "saddle" critical points. Kenwright et al. [Ken99] developed a technique to extract open separation lines using local phase-plane analysis. The approach successfully extracted both open and closed separation lines but had two drawbacks: it could not detect singular streamlines near vortices, and it was inaccurate in regions where the vector field was highly non-linear.

Much of the work in vector field topology employs linear expansions of the field in the neighborhood of critical points. However, this approach fails to capture the behavior of fields containing critical points with index other than 1 or –1. Scheuermann *et al.* introduced techniques to identify high-index critical points in 2-D [Sch98] although they did not address the issue of open and closed separation lines. Much of the existing work assumes a field domain of finite extent, whereas recent simulations of fluid flows using Lagrangian (particle based) methods can have domains of infinite extent.

For vector fields defined on an infinite domain, it is crucial to consider the "point at infinity" as an additional critical point [Nee97]. In 2-D, this idea has recently allowed us to find both open and closed separation lines by extending the ideas in vector field topology.

2. Vector field representation

To better understand such nonlinear vector fields, we have implemented a vector field designer based on the representation given by Scheuermann *et al.* [Sch98]. The user specifies the positions and topological indices of critical points in the plane, building up a Geometric Algebra polynomial vector field of the form:

$$\mathbf{u}(\mathbf{x}) = \mathbf{e}_1 e^{\mathbf{i}j} \prod_{j=1}^{N_1} (\mathbf{e}_1\mathbf{x} - z_j)^{n_j} \overline{\prod_{j=N_1+1}^{N_1+N_2} (\mathbf{e}_1\mathbf{x} - z_j)^{n_j}}$$

where: N_1 is the number of positive critical points; N_2 is the number of negative critical points; $\{\mathbf{e}_1, \mathbf{e}_2\}$ is an orthonormal basis for the plane; $\mathbf{i} = \mathbf{e}_1\mathbf{e}_2$, is the unit pseudoscalar (or unit imaginary number) of the plane; $\mathbf{x}_j$ is the location of the *j*'th critical point in the plane; $z_j = \mathbf{e}_1\mathbf{x}_j$ is the 2D spinor (or complex number) taking $\mathbf{e}_1$ to $\mathbf{x}_j$ (*i.e.*, $\mathbf{e}_1\mathbf{e}_1\mathbf{x}_j = \mathbf{x}_j$); n_j is the (integer) absolute value of the *j*'th critical point index; j is a user-specified angle by which every vector in $\mathbf{u}$ is rotated; and $\bar{z}$ denotes the conjugate of *z* for any 2D spinor *z*. This vector field representation causes numerical overflow for large values of $|\mathbf{x}|$, so it is necessary to find a more stable representation. Rewriting in polar coordinates, we obtain:

$$\mathbf{u}(\mathbf{x}) = \mathbf{e}_1 e^{\mathbf{i}j} \prod_{j=1}^{N_1} (r_j(\mathbf{x}))^{n_j} e^{\mathbf{i}n_j q_j(\mathbf{x})} \overline{\prod_{j=N_1+1}^{N_1+N_2} (r_j(\mathbf{x}))^{n_j} e^{\mathbf{i}n_j q_j(\mathbf{x})}}$$
$$a\ \mathbf{e}_1 e^{\mathbf{i}j}\, e^{\mathbf{i}\left(\sum_{j=1}^{N_1} n_j q_j(\mathbf{x}) - \sum_{j=N_1+1}^{N_1+N_2} n_j q_j(\mathbf{x})\right)}$$

where $q_j(\mathbf{x})$ denotes the angle in radians from $\mathbf{e}_1$ to $\mathbf{x} - \mathbf{x}_j$, and $r_j(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_j|$. The last formula is theoretically equivalent to normalizing all of the non-zero vectors of $\mathbf{u}$, which amounts to nothing more than re-parameterizing all of $\mathbf{u}$'s solution curves. This formula allows us to freely sample the vector field in large regions, without overflow. A comprehensive introduction to Geometric Algebra can be found in [Hes98].

3. Feature detection

The first phase of our feature detection algorithm involves integrating in critical directions from finite multisaddles (critical points having negative topological index), just as in [Sch98]. The second phase detects trajectories going to and coming from infinity by stereographically projecting the plane onto the unit sphere (Figure 3) and regarding the north pole as an additional critical point, the so-called *point at infinity* [Nee97, pp. 139-148].

The Poincare-Hopf theorem then asserts that:

$$I_\infty + I_T = 2 - 2g = 2 = c$$

where: I_∞ is the index of point at infinity; $I_T = \sum_{j=1}^{N_1} n_j - \sum_{j=N_1+1}^{N_1+N_2} n_j$ is the sum of all the indices of finite critical points; g=0 is the genus of the sphere; and c is the Euler characteristic of the sphere [Nee97, p. 464]. We may then solve for I_∞. If it is zero, then the vector field has a total index of two and behaves like a dipole when viewed from infinitely far away. We therefore integrate two critical trajectories "from infinity", starting at a large positive and a large negative multiple of $\mathbf{e}_1 e^{\mathbf{i}j}$. If I_∞ is negative, then the point at infinity is a multi-saddle and we integrate from large multiples of $\pm \mathbf{e}_1 e^{\mathbf{i}j_j}$, where j=0,…,-I_∞ and $j_j = (jp + j)/(-I_\infty)$. Otherwise, the point at infinity is a node, center, focus, or multipole (a critical point having index of at least two), so we have already found its critical trajectories by integrating from finite multisaddles.

4. Results

Using our geometric algebra vector field designer, several complicated 2-D vector fields were generated that contained both open and closed separation lines. One of these vector fields is illustrated in Figure 2. Singular streamlines originate from four finite critical points and one infinite critical point, i.e., the critical point at infinity. The attracting and repelling singular streamlines (separation and attachment lines in fluid dynamics) are colored green and red respectively. These are superimposed on normalized vector arrows that indicate the local direction of the vector field. The index of each critical point and their relationship to other critical points is illustrated in the topology graph in Figure 1.

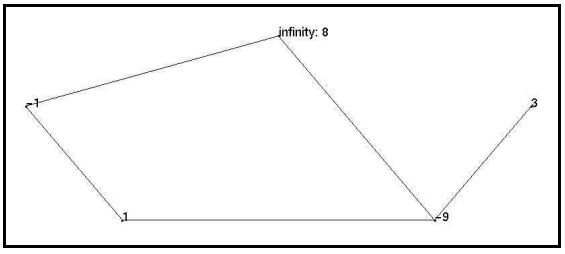


Figure 1. A topology graph that shows the index and relationship of every critical point in the test vector field.

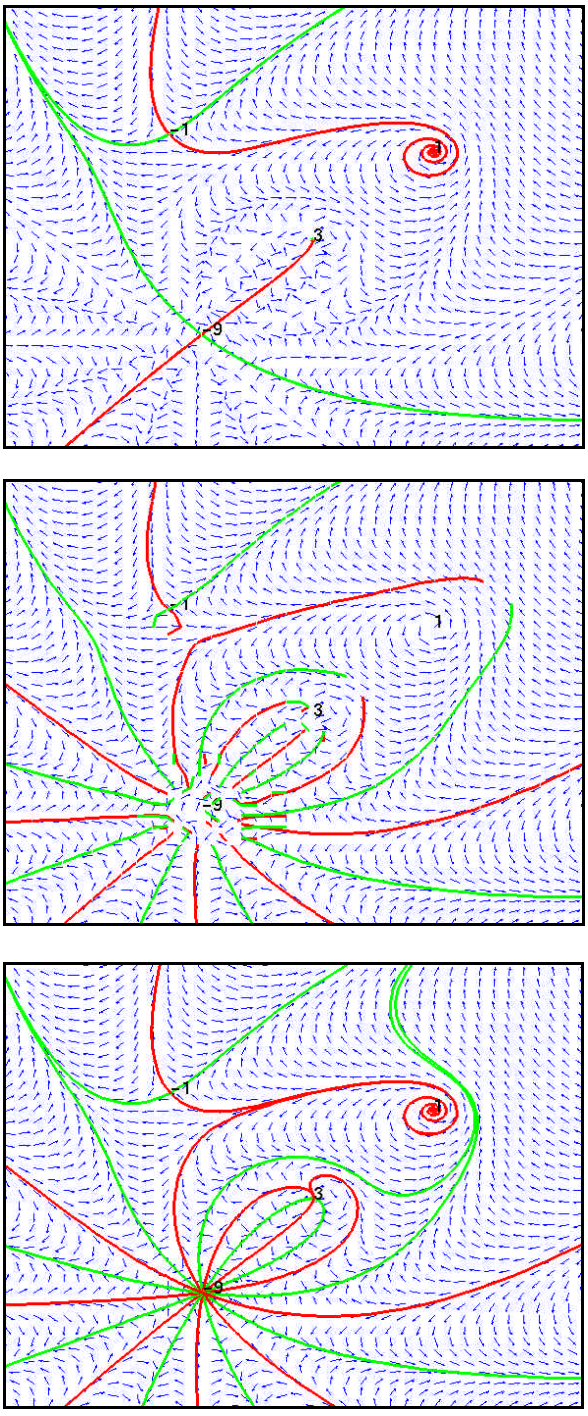


Figure 2. Singular streamlines extracted from a complicated 2D vector field containing low and high index critical points. Three techniques are shown: conventional vector field topology (upper), local phase plane analysis (center), and global vector field topology with critical point at infinity (lower).

Singular streamlines were extracted from the analytical vector field using one of three techniques. The first was linear vector field topology [Hel91], the second was local phase-plane analysis [Ken99], and the third was global vector field topology (Section 3). The results shown in Figure 2 exemplify the differences between these techniques. The linear vector field topology method only identifies singular streamlines that originate or terminate at saddle points within the computational domain. The local phase-plane analysis technique extracts considerably more lines including both open and closed singular streamlines. However, some those lines prematurely end as they approach spiral foci because the phase plane algorithm does not have a local definition for singular streamlines where the eigenvalues of $\nabla\mathbf{u}$ are complex numbers.

The global vector field topology shown in Figure 2, which includes singular streamlines originating from the critical point at infinity, produces a complete set of singular streamlines. The new algorithm clearly improves on the local phase-plane method because it is not limited to regions where the eigenvalues are real numbers. The 5^{th}-order Runge-Kutta-Fehlberg numerical integration scheme also produces more accurate results in non-linear regions than the analytical linear methods employed in the phase plane algorithm.

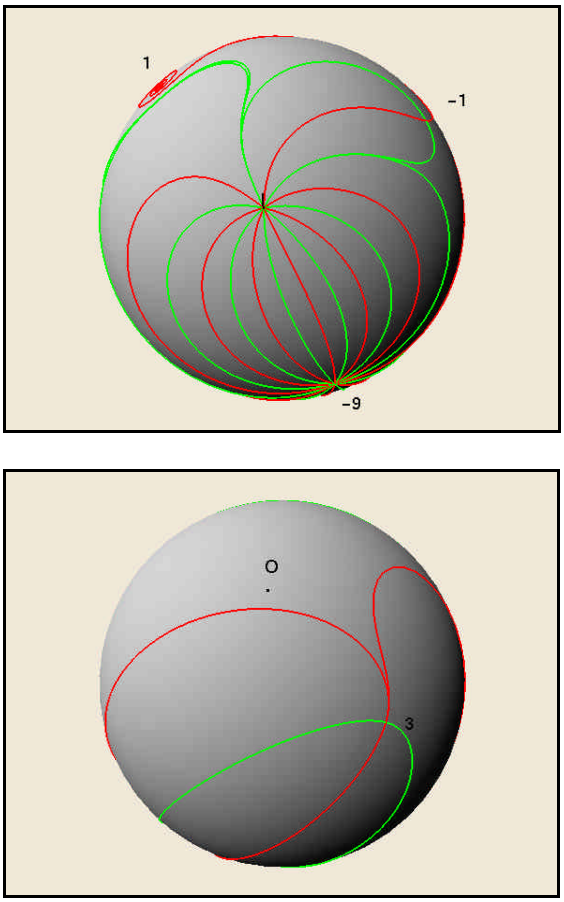


Figure 3. A stereographic projection of the vector field onto a unit sphere provides a "big picture" view of all singular lines that originate from infinity as well as those from finite critical points. Two views of the same sphere are shown to illustrate all five critical points. The singular point at infinity lies in the center of the sphere in the upper image.

5. References

[Glo91] A. Globus, C. Levit, T. Lasinski, *A tool for visualizing the topology of 3-D vector fields*. In Proceedings of IEEE Visualization '91, pp. 33-39, San Diego, CA, October 1991.

[Hai95] R. Haimes, Unsteady Visualization of Grand Challenge Size CFD Problems: Traditional Post-Processing vs. Co-Processing, Proceedings of ICASE/LeRC Symposium on Visualizing Time-Varying Data, Sept. 1995.

[Hel91] J. Helman and L. Hesselink, *Visualizing vector field topology in fluid flows*, IEEE Computer Graphics and Applications, Vol. 11, no. 3, pp. 36-46, May 1991.

[Hes98] D. Hestenes, Synopsis of Geometric Algebra, 1998. http://modelingnts.la.asu.edu/pdf/NFMPchapt1.pdf

[Ken99] D.N. Kenwright, C. Henze, and C. Levit, *Feature extraction of separation and attachment lines*, IEEE Transactions on Visualization and Comp-ter Graphics, IEEE Computer Society Press, vol. 5, no. 2, pp. 135-144, April-June 1999.

[Nee97] T. Needham, *Visual Complex Analysis*, Oxford University Press Inc., New York (1997).

[Sch98] G. Scheuermann, H. Kruger, M. Menzel, A.P. Rockwood, *Visualizing non-linear vector field topology*. In IEEE Transactions on Visualization and Computer Graphics, vol. 4, no. 2, pp. 109-116, 1998.

[Rot00] M. Roth, *Automatic Extraction of Vortex Core Lines and Other Line-Type Features for Scientific Visualization*, Ph.D. Dissertation, Swiss Federal Institute of Technology Zurich, May, 2000.

Simplification of Nonconvex Tetrahedral Meshes

Martin Kraus, Thomas Ertl

Visualization and Interactive Systems Group
Universität Stuttgart, Germany*

Abstract

We present a new solution to the well-known problems of edge collapses in nonconvex tetrahedral meshes. Additionally, our method is able to handle meshes with topologically non-trivial boundaries and to control the modification of the topology of the mesh's boundary.

1 Introduction

In order to visualize today's huge data sets, hierarchical representations are often employed. The production of such representations of tetrahedral volume meshes requires a simplification of the original mesh. One way of performing this simplification is to apply a sequence of edge collapses, which are discussed in some detail in section 2 with an emphasis on the problems of edge collapses in nonconvex meshes.

Our solution to these problems consists of two parts: A preprocessing step—originally suggested by Peter Williams in the context of sorting nonconvex meshes—is briefly described in section 3.

The second part—edge collapses in the resultant meshes—is discussed in section 4 with special attention being paid to modifications of the topology of the mesh's boundary.

Section 5 presents our conclusions and plans for future work on this subject.

2 Background and Related Work

2.1 Edge Collapses

In the following we will discuss edge collapses in fair tetrahedral meshes, i.e. each face of a tetrahedral cell is either part of the boundary of the mesh or shared by (at most) two cells. Intersections of cells are not allowed; in fact avoiding them is our primary concern. As edge collapses in triangular meshes in two dimensions are very similar to the three-dimensional case of tetrahedral meshes, we will illustrate our method with triangular meshes in Figures 1-8 before presenting the method in three dimensions in Figures 9-13.

It is useful for the description of our method to define some particular terms. We call a tetrahedron a *vertex neighbor* of a vertex if the vertex is shared by the tetrahedron. A tetrahedron is an *edge neighbor* of an edge if the edge is shared, and a *vertex neighbor* of an edge if one of the vertices of the edge is shared. Finally, a tetrahedron is a *face neighbor* of another tetrahedron if the tetrahedra share one face. The definitions of *vertex* and *edge neighbor*s can also be applied to triangles in triangular meshes.

The effect of an edge collapse (see Figure 1 for a two-dimensional example) is to remove all edge neighbors of the collapsing edge and to join the two vertices of the collapsing edge in a new vertex. Figure 1 also indicates the inverse operation, which is called a *vertex split*.

*Universität Stuttgart, IfI, Abt. VIS, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany; E-mail: {Martin.Kraus | Thomas.Ertl}@informatik.uni-stuttgart.de.

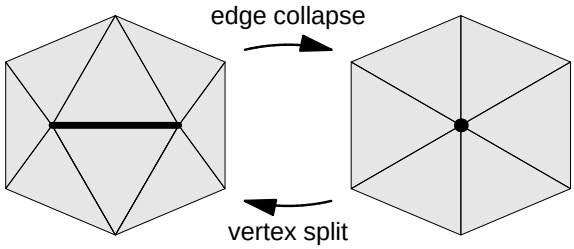


Figure 1: An edge collapse and the inverse vertex split.

Edge collapses are one of the most powerful tools to simplify triangular or tetrahedral meshes. They can be employed to remove vertices or edges (see [4]) and also to remove triangles or tetrahedra by successive edge collapses (see [5]).

Moreover, a sequence of edge collapses can be used to produce hierarchical representations of triangular and tetrahedral meshes as demonstrated in many publications, for example [3, 4, 5].

2.2 Avoiding Intersections of Cells

As demonstrated in Figure 2 an edge collapse can cause an intersection of cells in a triangular or tetrahedral mesh. In order to avoid such self-intersections of a mesh, edge collapses are tested before they are performed (see [4, 5]).

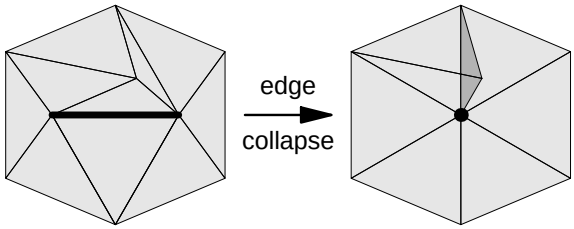


Figure 2: An edge collapse, which causes several intersections of cells and one inversion of a cell (dark gray).

In convex meshes, i.e. meshes the boundary of which are convex polytopes, the test for intersections is particularly simple because any intersection of cells is accompanied by an *inversion* of at least one cell, i.e. a sign flip of the signed volume of a cell. (The inverted cell is marked gray in Figure 2.) Therefore, it is sufficient to test all vertex neighbors of a collapsing edge for inversions in order to avoid self-intersections. This test is *local* as only vertex neighbors are involved.

However, if a collapsing edge in a nonconvex mesh has vertex neighbors that are cells at the boundary (i.e. one of the faces of the cell is part of the boundary of the mesh) then the edge collapse can cause self-intersections of the mesh without causing an inversion of a cell as shown in Figures 3 and 10.

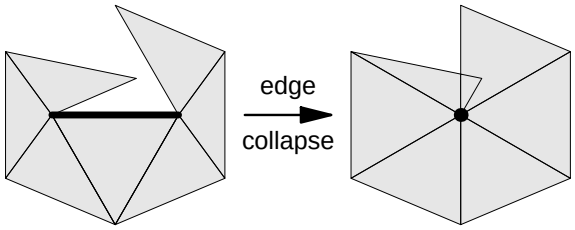


Figure 3: An edge collapse,which causes an intersection of two cells without causing an inversion of any cell.

A naive procedure to avoid such intersections is to test the vertex neighbors of the collapsing edge for intersections with all other cells of the mesh. As the time complexity of this *global* test depends linearly on the number of cells in the whole mesh, it is usually too expensive to be performed without additional auxiliary data structures. A more elaborated implementation of this test is discussed in [4] while the system described in [5] tries to preserve the boundary of the tetrahedral mesh.

However, performance issues are not the only problem of edge collapses in nonconvex meshes. Additional problems occur in disconnected meshes as edge collapses are obviously not able to join clusters of disconnected meshes in order to simplify them. More generally spoken, it is desirable to modify the topology of the mesh's boundary in a controlled way when performing edge collapses.

Before presenting our approach to solve these problems in section 4, we describe the necessary preprocessing step in the following section.

3 Convexification of Nonconvex Meshes

More than eight years ago Peter Williams proposed in [6] to convert nonconvex meshes to convex meshes by triangulating all voids and cavities and marking the cells generated by this triangulation as *imaginary*. (*Virtual* is today's more fashionable word for the same idea.)

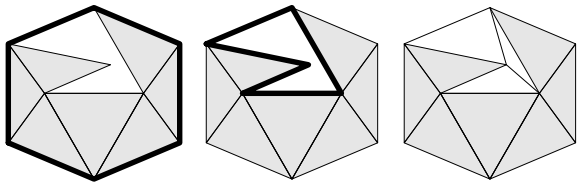


Figure 4: A step-by-step convexification of the mesh shown in the left-hand side of Figure 3. From left to right: the convex hull (thick line), the nonconvex polygon (thick line) between the convex hull and the boundary of the mesh, and the mesh together with imaginary cells (white) generated by the triangulation of the nonconvex polygon.

Figures 4 and 11 summarize the basic steps of this process. Firstly, the convex hull of the mesh is computed; then all voids and cavities are identified and triangulated; finally, the new imaginary cells are attached to the existing mesh. (As will be shown in section 4 the number of imaginary cells generated by the triangulation is not relevant in the context of edge collapses. An optimal algorithm (with respect to the number of generated tetrahedra) for the tetrahedralization of nonconvex polyhedra was published in [1].)

We call this preprocessing step a *convexification* of a nonconvex mesh as it allows us to apply (slightly modified) algorithms for convex meshes to a nonconvex mesh. (In this sense the convexification might be called a *meta-algorithm*.) The following section shows how to overcome problems of edge collapses introduced by nonconvexities (including non-trivial topologies of the boundary) with the help of this preprocessing step.

4 Simplification of Convexified Meshes

4.1 Geometric Tests

As mentioned in section 2 and shown in Figure 3, edge collapses in nonconvex meshes can cause intersections of cells without causing an inversion of any cell. In convexified meshes, however, such edge collapses will always cause an inversion of at least one imaginary cell as shown in Figure 5 for the same edge collapse as in Figure 3 in a convexified version of the same triangular mesh.

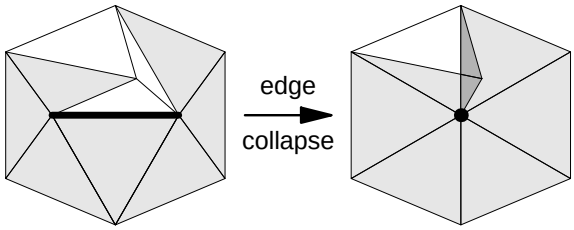


Figure 5: The edge collapse of Figure 3 in the convexified mesh from Figure 4 causes an inversion of an imaginary cell (dark gray). (Compare also with Figure 2.)

Therefore, convexified meshes allow us to test for self-intersections of cells by simply testing all vertex neighbors (including imaginary cells) of the collapsing edge for sign flips of the signed cell volume, which is a local, geometric test as in the case of convex meshes. Thus, the total number of new imaginary cells generated by the convexification is not relevant for the efficiency of this test.

4.2 Preservation of the Convex Hull

Not only are edge collapses in nonconvex meshes more complicated than in convex meshes, they can also transform a convex mesh into a nonconvex mesh.

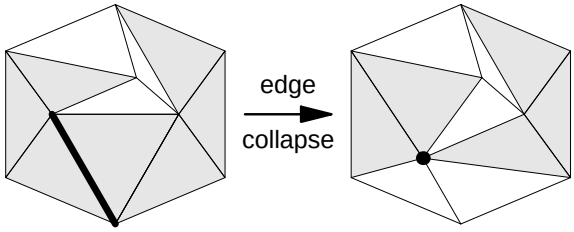


Figure 6: An edge collapse, which could generate a nonconvexity. Two new imaginary cells are inserted in order to preserve the original convex hull.

An example is depicted in Figure 6, which also shows our solution: Instead of recomputing the convex hull (a global operation if implemented naively), we insert imaginary cells between the new vertex and the convex hull in order to preserve the original convex hull. (The effect can also be seen at the bottom of Figure 12.)

This is an efficient, local operation. However, it will also insert some new edges; therefore, a simplification process might run into an endless loop by collapsing edges which are instantly reconstructed by the insertion of imaginary cells. In order to avoid this problem, a simple test for edge collapses has to be added. Edge collapses are avoided if the following three conditions are met: All edge neighbors of the collapsing edge are imaginary, one vertex is part of the boundary of the mesh, and all vertex neighbors of this vertex are imaginary. (For an example see the edge between the new imaginary cells in the right-hand side of Figure 6.)

4.3 Topology Preservation

Edge collapses in convexified meshes are considerably more powerful than edge collapses in the original meshes. For example, disconnected meshes can be joined in order to be simplified, tunnels in the boundary of a mesh can be closed, bridges between meshes can be broken, etc. (Figure 12 shows a new connection between originally disconnected parts of the mesh in the top-right corner and a disconnection of cells at the bottom.)

However, not all of these features are always welcome; instead, it is more appropriate to have full control over the modifications of the topology of the mesh's boundary. Here we present two very simple tests for edge collapses in order to avoid topological changes of the mesh. Both tests involve only vertex neighbors of the collapsing edge. Before presenting these topological tests, we have to define some basic terms.

Def.: The *type* of a cell is either imaginary or non-imaginary.

Def.: A cell T_1 is *connected* to a cell T_2 of the same type if the cells share a vertex (direct connection), or if T_1 is connected to a third cell T_3 of the same type that is connected to T_2 (indirect connection).

Def.: Two cells are *disconnected* if they are not connected.

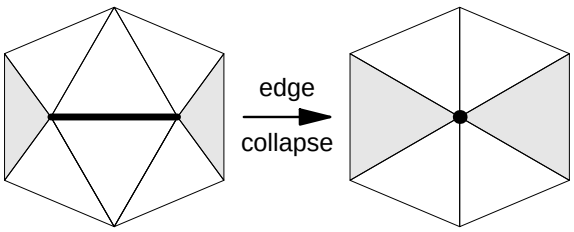


Figure 7: An edge collapse, which connects two non-imaginary cells (gray).

Figure 7 shows an example of an edge collapse that establishes a new connection between two non-imaginary cells. In general, the collapse of an edge e between two vertices v_1 and v_2 *can* connect two previously disconnected cells if all of the following three conditions are met: All of the edge neighbors of e are of the same type t; at least one of the vertex neighbors of v_1 is not of type t; and at least one of the vertex neighbors of v_2 is not of type t. This is a necessary condition; therefore, it is sufficient to avoid edge collapses that fulfill it in order to avoid new connections between cells. The test is the same for triangular and tetrahedral meshes.

Edge collapses are also able to disconnect cells; an example is presented in Figure 8. In order to formulate a test for disconnecting edge collapses, we need one more definition:

Def.: An edge neighbor T of type t of an edge e is *isolated* if none of the faces of T that do not share e are shared by a face neighbor of T of type t.

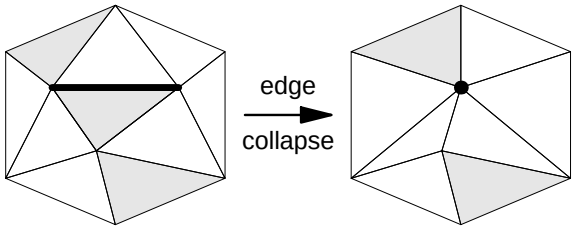


Figure 8: An edge collapse, which disconnects two non-imaginary cells (gray).

Using this definition we can state that the collapse of an edge e *can* disconnect two previously connected cells if at least one of the edge neighbors of e is isolated. This is again a necessary condition, which works for triangular and tetrahedral meshes. (Note that the faces of a triangular cell are its edges.)

These two topological tests allow us to avoid new connections and/or disconnections simply by avoiding edge collapses that fulfill the conditions stated above. An example with a topological nontrival mesh is given in Figure 13, which should be compared with Figure 12, where these tests were not applied.

5 Conclusions and Future Work

An idea of Peter Williams to convert nonconvex into convex meshes was successfully applied to solve the problems of edge collapses in nonconvex tetrahedral meshes with topologically non-trival boundaries, which may be non-manifolds. Intersections of cells, new nonconvexities of the boundary, and modifications of the topology are identified and avoided by efficient, local tests. Therefore, the particular problems of simplifying nonconvex meshes by edge collapses are in principle solved.

However, many problems are still open: an efficient and robust computation of the convex hull of an arbitrary tetrahedral mesh; an efficient and robust computation of the tetrahedralization of an arbitrary polyhedron; more elaborated definitions of topology preserving edge collapses (see [2]); and, of course, applications to real-world data sets.

References

[1] Bernard Chazelle and Leonidas Palios. Triangulating a Nonconvex Polytope. *Discrete & Computational Geometry*, 5:505-526, 1990.

[2] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology Preserving Edge Contraction. *Publ. Inst. Math. (Beograd) (N.S.)*, 66:23-45, 1999.

[3] Jovan Popović and Hugues Hoppe. Progressive Simplicial Complexes. In *Computer Graphics Proceedings (SIGGRAPH '97)*, pages 217-224, 1997.

[4] Oliver G. Staadt and Markus H. Gross. Progressive Tetrahedralizations. In *Proceedings of IEEE Visualization '98*, pages 397-402, 1998.

[5] Issac J. Trotts, Bernd Hamann, Kenneth I. Joy, and David F. Wiley. Simplification of Tetrahedral Meshes. In *Proceedings of IEEE Visualization '98*, pages 287-295, 1998.

[6] Peter L. Williams. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics*, 11(2):103-126, 1992.

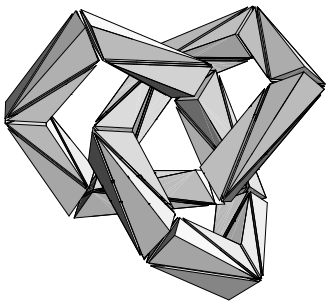


Figure 9: The nonconvex tetrahedral mesh that is the starting point for the calculations depicted in Figures 10-13.

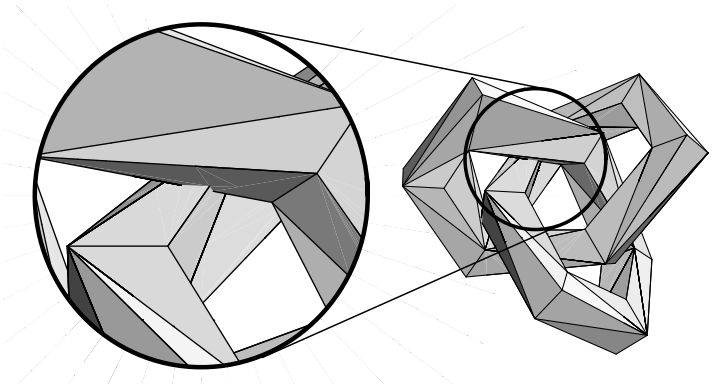


Figure 10: The tetrahedral mesh from Figure 9 after one edge collapse, which causes two self-intersections of the mesh. One of the self-intersections is shown in detail.

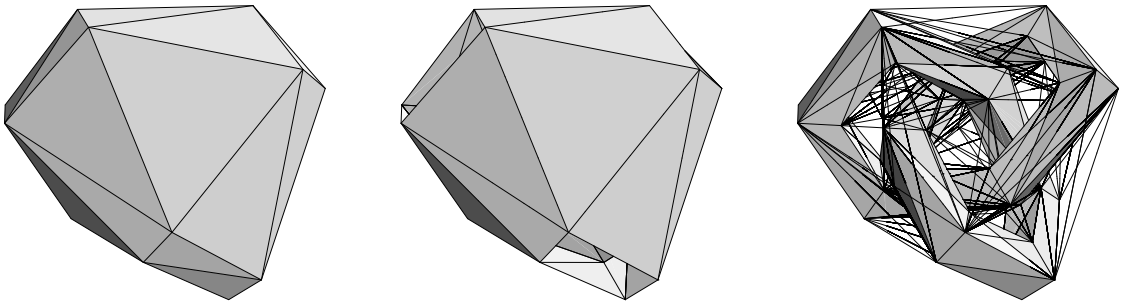


Figure 11: (Intermediate) results of the convexification of the mesh shown in Figure 9. From left to right: the convex hull of the mesh; the nonconvex polyhedron between the convex hull and the boundary of the mesh, which has to be tetrahedralized; and the mesh together with imaginary tetrahedra generated by the tetrahedralization (only edges of imaginary cells are shown).

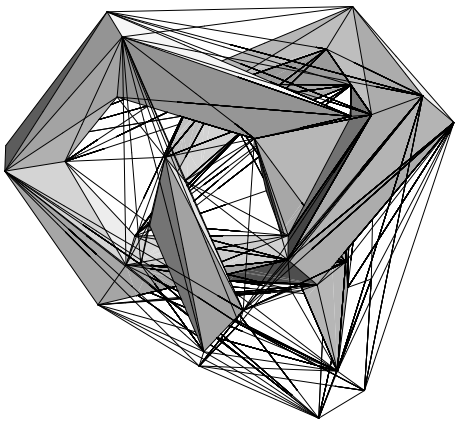


Figure 12: The result of a simplification of the convexified mesh depicted in Figure 11 without topological tests. The geometric tests guarantee that there are no self-intersections and hamper further edge collapses.

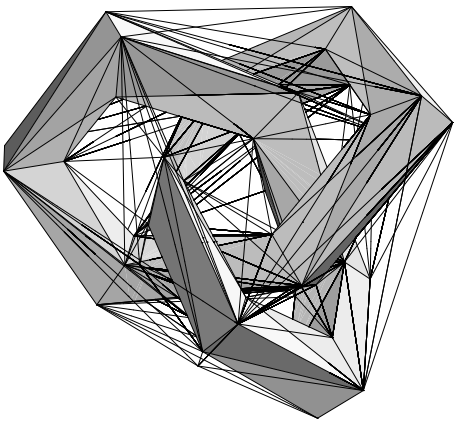


Figure 13: Topological tests in a simplification of the mesh from Figure 11 guarantee the preservation of the connectivity. Therefore, the simplification process is halted earlier than without these tests.

A Multi-Resolution Interactive Previewer for Volumetric Data on Arbitrary Meshes

Oliver Kreylos* Kwan-Liu Ma* Bernd Hamann*

Abstract

In this paper we describe a rendering method suitable for interactive previewing of large-scale arbitary-mesh volume data sets. A data set to be visualized is represented by a "point cloud," i. e., a set of points and associated data values without known connectivity between the points. The method uses a multi-resolution approach to achieve interactive rendering rates of several frames per second for arbitrarily large data sets. Lower-resolution approximations of an original data set are created by iteratively applying a point-decimation operation to higher-resolution levels. The goal of this method is to provide the user with an interactive navigation and exploration tool to determine good viewpoints and transfer functions to pass on to a high-quality volume renderer that uses a standard algorithm.

1 Introduction

Current visualization methods for arbitrary-mesh, volumetric data sets do not allow interactive rendering, or even low-quality previewing, of large-scale data sets containing several million grid points. In most cases, a scientist creates or measures such a data set without a-priori knowledge of where to find the features she is looking for; sometimes, even without knowing what those features are. Volume visualization has proven to be a very helpful tool in these situations. But without interactive navigation and exploration tools, finding features in a very large data set and highlighting them using customized transfer functions is very difficult and time-consuming.

If images of a data set could somehow be rendered at interactive rates, even at relatively poor quality, the navigation process could be sped up considerably.

1.1 Related Work

There are several basic rendering methods for arbitrary-mesh volumetric data sets that are geared towards generating high-quality images at the expense of rendering time. These methods include the ray casting algorithm described by Garrity [2], the cell projection algorithm discussed by Lichan and Kaufman [5], the plane-sweep modification of the ray casting algorithm invented by Silva et al. [7], the adaptation of the splatting algorithm for non-rectilinear volumes developed by Mao [3], the polygonal approximation to ray casting presented by Shirley and Tuchman [4], and the slicing approach described by Yagel et al. [6].

Researchers have tried optimizing these algorithms following different approaches. Probably the easiest optimization is subsampling in image space, by generating small images and duplicating pixels using some reconstruction filter. A more sophisticated approach is utilizing graphics hardware for volume rendering. This has been a major success for rectilinear data sets, where 3D texture mapping can be used to generate images at interactive rates [8]. Yagel et al. [6] developed a similar method that generates slices of tetrahedral mesh data sets and uses hardware-assisted polygon rendering to generate images of and composite these slices. There has also been a considerable amount of work on utilizing massively parallel supercomputers to speed up volume rendering [1, 9, 11].

*Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, One Shields Avenue, Davis, CA 95616–8562, {kreylos,ma,hamann}@cs.ucdavis.edu

1.2 Interactive Previewing of Large-Scale Volume Data Sets

We describe a new rendering method for irregular volume data sets that uses multiresolution approximations to trade off image quality against rendering speed. This method does not use the topology information contained in irregular data sets, but attempts to reconstruct images of a data set by looking at the data values at grid vertices only. Obviously, this method only generates approximations, but experiments show that the quality of the generated images, combined with the fact that these images are generated rapidly, is more than sufficient to allow the user to detect and highlight features in a data set quickly, see section 5. After good viewpoints and transfer functions have been determined in the previewing phase, those are passed on to either a high-accuracy rendering method [10] or a high-performance rendering method [11].

1.3 Throwing Away the Topology

To allow rapid rendering of approximations of an arbitrary-mesh data set, our algorithm does not take the topology of a given grid into account. Instead, it treats the data set as a cloud of points (with associated data values) without known connectivity. Of course, doing so radically decreases the image quality: without knowledge of the vertex connectivity, any rendering can only be an approximation of the correct image. On the other hand, rendering a point cloud has the following benefits:

1. Since it is only an approximation to begin with, one can select a convenient approximation method that utilizes graphics hardware.
2. The algorithm described in section 2 can easily be parallelized for shared-memory, multi-processor graphics workstations.
3. It is comparatively easy to decimate a point cloud to generate a hierarchy of approximations at multiple levels of resolution.

Using these optimizations, and selecting the appropriate hierarchy level for the user's demands, allows to create an

algorithm that renders approximations of arbitrarily large data sets at interactive frame rates.

2 Point-Based Volume Rendering

The major problem of point-based volume rendering is to generate a continuous image. Rendering all points in the set independently, e.g. using a splatting method, usually does not work. In many irregular data sets, the distances between neighbouring points vary over several orders of magnitude; drawing the point cloud with a fixed-size splatting kernel would induce holes in the image in sparse regions and overpainting in dense regions of the data set.

Using variable-shape splatting kernels could solve the problem, but finding out the correct shape to use for a given point is a major task in itself when the connectivity of the points is unknown.

2.1 Rendering a Point Cloud

Our algorithm follows the following basic idea to "fill in" pixel values between neighbouring points:

1. A given point set is transformed such that the viewing direction is along the negative z-axis. This step, called "transformation," is an additional step to optimize later stages of the algorithm.

2. The point set is subdivided into thin "slabs" that are orthogonal to the viewing direction, i. e., each slab is of nearly constant z value. We refer to this step as "slicing." Slicing is done adaptively to take the varying point density in a data set into account.

3. The slabs are converted into a continuous representation (a triangle mesh) by creating the Delaunay triangulation of all points included in the slab. We call this step "meshing."

4. The meshes associated with each slab are rendered and composited in back-to-front order using hardware-accelerated polygon rendering and alpha blending. This step is appropriately called "rendering."

These four steps describe a four-stage rendering pipeline, shown in Figure 1.

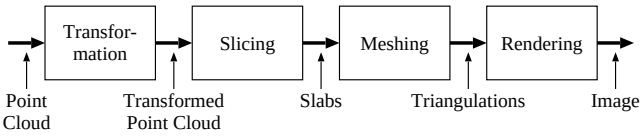


Figure 1: The four-stage rendering pipeline defined by our algorithm.

2.2 Visible Artifacts

The major cause of visible artifacts in resulting images is the fact that each of the slabs generated by the slicing process is triangulated and rendered independently. This can lead to the effect that the image of one triangulation is not influenced by a point that is very close to the triangulation in object space, but happens to be inside a different slab, see Figure 2.

Since the two slabs depicted in Figure 2 are rendered independently, the color and opacity values are interpolated linearly between points P_1 and P_2. In a correct rendering, the values would have to be interpolated between points P_1 and P_3, and then between points P_3 and P_2. But, because point P_3 is located in a different slab, the algorithm is oblivious to this fact.

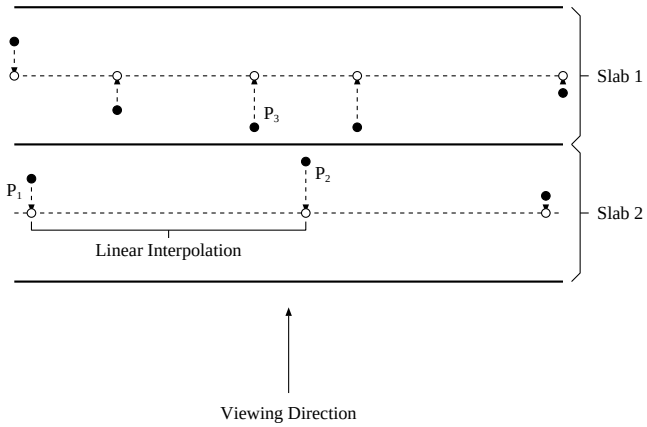


Figure 2: Potential artifacts in resulting images. Inside slab 2, color and opacity values are wrongly interpolated between points P_1 and P_2.

These artifacts are especially visible when a slab contains only a small number of points, or when all points are clustered in a small region of the triangulation's intersection with the bounding box of the point set. In these cases, the meshing process connects the points to the vertices on the triangulation's boundary, and the resulting long and thin triangles will "smear out" the color values all the way to the boundary.

The distinct appearance of these visual artifacts is, in some sense, beneficial: it is hard to misinterpret them as features in a data set. Since detecting and emphasizing features present in a data set is the major goal of our algorithm, it is usable in spite of these distortions. The images produced are not intended to be used "as is," but they provide help in navigating through a large data set, and in finding interesting viewpoints and transfer functions to pass on for subsequent high-quality renderering.

3 Parallel Rendering

The serial implementation of our algorithm, as described in section 2, is already capable of rendering small data sets (consisting of several thousand points) on a standard graphics workstation, e. g., an SGI O2, at interactive rates of several frames per second, see section 5.

To improve the efficiency of the algorithm, we decided to parallelize it for use on multi-processor shared-memory graphics workstations, like SGI Onyx2 workstations. To distribute the workload among the processors, we exploit both functional parallelism inside the rendering pipeline and object-space parallelism.

3.1 Functional Parallelism

To exploit functional parallelism, we decouple the rendering pipeline as shown in Figure 1 by creating separate threads for each stage and connecting the stages by request queues.

The first pipeline stage is handled by a single thread, because it requires only a very short amount of time, and parallelizing it would incur too much overhead. The second and third pipeline stages are represented by a pool of worker threads. The final pipeline stage is also done by a single thread, because the triangulated slices have to be rendered in order, and the OpenGL implementation available to us does not support concurrent rendering into a single frame buffer. The overall structure of our parallel pipeline is shown in Figure 3.

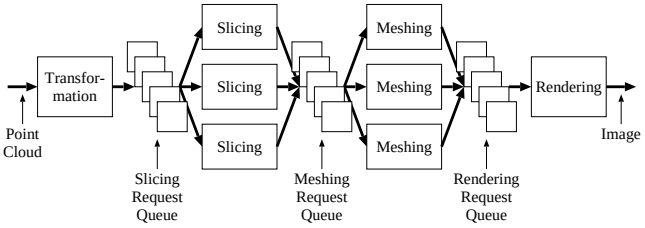


Figure 3: The parallel rendering pipeline defined by our algorithm.

One detail of our parellel rendering pipeline is not shown in Figure 3: in order to parallelize the inherently recursive slicing process, a slicer thread can also place a slicing request to the slice request queue. If a slicer thread determines that a slab is thin enough or contains few enough points to render, it will put an entry into the meshing request queue. If, on the other hand, the slab has to be subdivided further, it splits the slab and places two new slicing requests, one for each of the two generated slabs, to the slicing request queue. By following this strategy, we achieve good load balancing between the slicing threads.

3.2 Object-Space Parallelism

The threads in the slicing and meshing pipeline stages operate independently of each other. Therefore, we achieve object-space parallelism: as soon as a slab is subdivided into a "front" and a "back" portion, those can be processed in parallel. In the meshing process, all slices are independent of each other and can be created in parallel.

3.3 Comparison with the Serial Algorithm

As to be expected, the parallel version of our algorithm is considerably faster than the serial version when executed on a shared-memory multi-processor graphics workstation. We have compared the runtimes on a four-processor SGI Onyx2 workstation, and the parallel algorithm cuts down rendering time by a factor of about four, yielding a parallel efficiency of about 90%.

It is more surprising that even on a single-processor workstation the parallel version is faster than the serial one. We believe that the multi-threaded version overlaps the processes of mesh creation and mesh rendering. The latter is done completely in hardware, and in the serial version the CPU has to wait for the graphics subsystem to finish rendering, whereas the parallel version can continue to work in the slicing or meshing pipeline stages.

4 Multiresolution Rendering

Even after having parallelized the volume renderer, it is still not fast enough to render very large data sets containing millions of points. The reason for this is that the methods used in parallelization do not scale well beyond small numbers of processors in a shared-memory system.

To achieve our goal of interactive rendering of very large data sets, we have to create smaller approximations to those data sets first and render them instead. When creating a multiresolution approximation hierarchy, the program (or the user) can always specify an appropriate resolution level to trade off image quality against rendering time.

4.1 Creating a Hierarchy of Approximations

To create a hierarchy of approximations, we start with the point set of the original data set (and call it level 0) and perform a point decimation algorithm. We call the result level 1 and repeat the decimation algorithm for level i to generate level $(i + 1)$, and so forth. This process terminates when the result of the decimation algorithm is a sufficiently small data set. With current computer performance and interactive rendering in mind, "small" means that the coarsest-resolution level should contain only a few thousand points.

4.2 The Decimation Algorithm

Finding a sufficiently good representation of a given point set using only a fraction of the original points is difficult, especially when the original points are not aligned on a regular grid. In that case, the algorithm could just sub-sample the grid (by only choosing every other point) and would generate a meaningful approximation (modulo aliasing).

In the case of arbitrary-mesh data sets, points are not "aligned" and generally do not form a lattice that could be sub-sampled easily. Even worse, point density might vary over several orders of magnitude in a single data set.

Therefore, we need an algorithm that resembles the sub-sampling approach for regular grids, in the sense that it keeps the relative point densities of an approximation similar to the relative point densities of the original data set.

The algorithm we chose to preserve point densities is based on *maximum independent sets*. To create an approximation, we first calculate a Delaunay tetrahedrization of the original data set. This results in a tetrahedral mesh where each point is connected to all its nearest neighbours by an edge.

As a second step we invoke a "mark-and-sweep" algorithm that extracts the maximum set of points such that no two points in the set are direct neighbours of each other in the original data set.

5 Examples and Results

We have applied our algorithm to several data sets of different sizes and recorded the runtimes for each data set. The original images generated by our algorithm can also be found under the URL
`http://graphics.cs.ucdavis.edu/~okreylos/Research/VolumeRendering/index.html`.

5.1 Measurements

Table 1 lists the rendering times for the parallel implementation of our algorithm, executed on an SGI Onyx2 workstation having four MIPS R10K processors running at 195 MHz and 512 MB of main memory. The rendered data sets are the ones described on the web page.

Dataset	# of points	time (sec.)
Mavriplis	438	0.01
	2,800	0.02
Parikh	378	0.01
	2,425	0.02
	15,804	0.12
	103,064	0.99
Shalf	6,607	0.05
	46,261	0.31
	346,087	2.66
	2,531,452	27.35

Table 1: Rendering times for various data sets.

6 Conclusion

To evaluate our point-based rendering method for arbitrary-mesh volumetric data sets, we have implemented an experimental application that allows navigating such data sets and creating colour and opacity maps to pass on to other volume rendering programs. Our multi-resolution approximation technique allows rendering approximations of data sets of varying sizes at interactive frame rates on a four-processor SGI Onyx2 graphics workstation.

In our experiments, we found that the rapid rendering achieved by our approach and implementation is a valuable help in finding and highlighting interesting features in an unknown data set quickly. The artifacts described in section 2.2 are visible, especially when rendering low-resolution approximations, but do not hinder the navigation process. As long as final images are generated by a standard high-quality volume rendering algorithm, the image distortions induced by our method are of little concern.

7 Acknowledgements

This work was supported by the National Science Foundation under contracts ACI 9624034 and ACI 9983641 (CAREER Awards), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251.

We thank the members of the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

We thank Dimitri Mavriplis at ICASE, Paresh Parikh at ViGYAN, Inc. and Greg Bryan, Mike Norman and John Shalf at the Laboratory for Computational Astrophysics at NCSA and Lawrence Berkeley National Laboratory for providing the data sets used as examples. We thank Gunther Weber at CIPIC for help with the AMR file format.

References

[1] Challinger, J., *Scalable Parallel Volume Ray-Casting for Nonrectilinear Computational Grids*, in Proc. 1993 Parallel Rendering Symposium (1993), ACM Press, pp. 81–88

[2] Garrity, M. P., *Raytracing Irregular Volume Data*, in Proc. 1990 Workshop on Volume Visualization, special issue of Computer Graphics, vol. 24(5) (1990), pp. 35–40

[3] Mao, X., *Splatting of Non-Rectilinear Volumes Through Stochastic Resampling*, in IEEE Transactions on Visualization and Computer Graphics, vol. 2(2) (1996), pp. 156–170

[4] Shirley, P. and Tuchman, A., *A Polygon Approximation to Direct Scalar Volume Rendering*, in Proc. 1990 Workshop on Volume Visualization, special issue of Computer Graphics, vol. 24(5) (1990), pp. 63–70

[5] Lichan, H. and Kaufman, A. E., *Fast Projection-Based Ray-Casting Algorithm for Rendering Curvilinear Volumes*, in IEEE Transactions on Visualization and Computer Graphics, vol. 5(4) (1999), pp. 322-332

[6] Yagel, R., Reed, D. M., Law, A., Shih, P. and Shareef, N., *Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing*, Proc. 1996 Volume Visualization Symposium, ACM SIGGRAPH (1996), pp. 55-62

[7] Silva, C. T., Mitchell, J. S. B. and Kaufman, A. E., *Fast Rendering of Irregular Volume Data*, in Proc. 1996 Volume Visualization Symposium, ACM SIGGRAPH (1996), pp. 15–22

[8] Meissner, M., Hoffmann, U. and Strasser, W., *Volume Rendering Using OpenGL and Extensions*, in Proc. Visualization '99, pp. 207–526

[9] Williams, P. L., *Parallel Volume Rendering Finite Element Data*, Proc. Computer Graphics International '93, Lausanne, Switzerland, June 1993

[10] Williams, P. L., Max, N. L. and Stein, C. M., *A High Accuracy Volume Renderer for Unstructured Data*, in IEEE Transactions on Visualization and Computer Graphics, vol. 4(1) (1998), pp. 37–54

[11] Ma, K.-L. and Crockett, T. W., *A Scalable Parallel Cell-Projection Volume Rendering Algorithm for Three-Dimensional Unstructured Data*, in Proc. IEEE Symposium on Parallel Rendering, IEEE Computer Society Press (1997),pp. 95–104

[12] Guibas, L. J., Knuth, D. E., and Sharir, M. *Randomized Incremental Construction of Delaunay and Voronoï Diagrams*, in Proc. 17th Int. Colloq.—Automata, Languages and Programming, vol. 443 of Springer Verlag LNCS (1990), Springer Verlag, Berlin, pp. 414–431

Efficient Error Calculation for Multiresolution Volume Visualization

Eric LaMar,
Bernd Hamann, and Kenneth I. Joy

October 3, 2000

Abstract

Multiresolution volume visualization is necessary to enable interactive rendering of large data sets. Interactive manipulation of the transfer function is necessary for proper exploration of a data set. However, multiresolution techniques require assessing the accuracy of the resulting images, and re-computing the error after each change in the transfer function is very expensive. We introduce a method for accelerating error calculations for multiresolution volume approximations. For computing error on byte data on large data sets, we observe that there is a large set of pairs of error terms, but there is a small set of unique pairs of error terms. Instead of evaluating the error function for each original voxel, we construct a table of the unique error terms and their frequencies. To evaluate the error, we compute the error function for each unique error term and multiply it by the frequency of that term. This method has a small, one-time cost, dramatically reduces the amount of computation involved in computing the error associated with a transfer function, and allows us to re-compute the error associated with a new transfer function quickly.

1 Introduction

When rendering images from approximations, is it often necessary to know how close the generated image is to the original data. For multiresolution volume visualization, it is not possible to compare the images generated from original data to all possible images generated from approximations. The reason for using the approximations is to substantially reduce the amount of time it takes to render the data. If we assume that there is a reasonable amount of correlation between the data and the resulting imagery, we can compute and error value between the approximations and the original data (in 3D "object space"), and use that to estimate the error in the 2D imagery.

However, the time required to evaluate the error over an entire data hierarchy can be very expensive. This is especially important when it is necessary for a user to interactively modify the transfer function: each change in the transfer function requires re-computing the error for the entire hierarchy. We introduce a solution based on the observation that, for many data sets, the range size of data sets is often many orders smaller that the domain size (physical extension).

Instead of evaluating an error function for each original voxel, we count the frequencies of unique pairs of error terms – and, for byte data, there are only 256^2 combinations (each term is a single byte, or 256 potentially different values). To compute the error, instead of adding individual error terms, we add the products of the error (for a unique pair of error terms) by the frequency of that unique pair of error terms.

2 Generating the Hierarchy

First, we review certain aspects of our multiresolution data representation and how it influences the decisions on how to evaluate and store error. The underlying assumption is that data sets are too large to fit into texture memory, and are often too large to even fit into main memory. Given a volumetric data set, we produce a hierarchy of approximations. Each level in the approximation hierarchy is half the size of the next level. Each level is broken into constant-sized tiles – tiles that are small enough to fit in their entirety in texture memory.

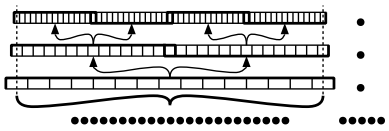


Figure 1: Texture Hierarchy of Three Levels.

Figure 1 illustrates a 1D multiresolution hierarchy of three levels and seven nodes, of a one-dimensional function with tile size of 15 texels. Each node in the tree contains one tile. Level 0 is the original data and is broken into four tiles. Level 1 is the first approximation and is broken into two tiles; and Level 2 is the final and coarsest approximation (of Level 0) and contains just one tile. The coarsest tile is contained in the root of the tree; the arrows show the parent-child relationship of the nodes. This structure forms a binary tree (and, in three-dimensions, an octree), and a Level i texel approximates $(2^i + 1)^d$ level 0 texels, where $d = 1$ for a binary tree, see Figure 1, and $d = 3$ for an octree.

3 Error Calculation

We calculate error on a per-node basis: when a node meets the error criterion, it is rendered; otherwise, its child (higher-resolution) nodes are considered for rendering. We currently assume a piece-wise constant function implied by a set of voxels, but use trilinear interpolation for the texture. This simplifies the error calculation.

We use two error norms: L-infinity and root-mean-squared (RMS). Given two sets of function values, $\{f_i\}$ and $\{g_i\}$, $i = 0, \ldots, n-1$, the L-infinity error norm is defined as $l_\infty = max_i \{|x_i|\}$, and the RMS is defined as $E_{rms} = \sqrt{\frac{1}{n}\sum_i (x_i^2)}$, where $x_i = T[f_i] - T[g_i]$ and $T[x]$ is the transfer function. Our system implements $T[x]$ with a look-up table, mapping density values to an RGBA (red, green, blue, alpha) tuple. We evaluate the error function once for each Level 0 voxel.

A data set of size 512^3 contains 2^{27} voxels, with the same number of pairs of error terms for each level of the hierarchy. However, when using byte data, we observe that, though there are 2^{27}

pairs of values, there are only 2^{16} (256^2) unique pairs of values. This means, on average, that each unique pair is evaluated 2^{11} times. Our solution is to build a table, with elements $Q_{f,g}$, that stores the frequency of each (f_i, g_i) pair, thus $Q_{f,g} = \sum_i \begin{pmatrix} 1 & f = f_i \,\&\, g = g_i \\ 0 & \text{else} \end{pmatrix}$. This table is created only once, when the data is loaded. Each internal node of the octree, i.e., each approximating node, contains a $Q_{f,g}$ table, that counts the unique error terms that node "covers" of the original data. We define $E_{f,g} = T[f_i] - T[g_i]$, and note that f_i and g_i refer to the function sets and the subscripted f and g refer to the corresponding table index. To calculate the L-infinity error value for a node, we search the table $E_{f,g}$ for the largest value, with the requirement that this value corresponds to a real pair of values $l_\infty = max_{f,g} \{|E_{f,g}|\} \vee Q_{f,g} \neq 0$. We compute the RMS error for a node as $E_{rms} = \sqrt{\frac{1}{n} \sum_{f,g} (E_{f,g})^2 \times Q_{f,g}}$, where $n = \sum_{f,g} (Q_{f,g})$. For a set of t nodes, we compute the value $E_{rms} = \sqrt{\frac{1}{N} \sum_t \left(\sum_{f,g} (E_{f,g})^2 \times Q_{f,g} \right)}$, where $N = \sum_t \left(\sum_{f,g} Q_{f,g} \right)$, and N is the total number of voxels in the original data set.

4 Optimizations

Evaluating a table is still fairly expensive, when interactive performance is required. We currently use three methods to reduce the time to evaluate the error.

First, one observes that the order of the error term calculations does not matter: $x_i = |f_i - g_i| = |g_i - f_i|$ and $x_i = (f_i - g_i)^2 = (g_i - f_i)^2$. If we order the indices in $Q_{f,g}$ such that $f \leq g$, we obtain a triangular matrix that has slightly more than half the terms of a full matrix (32896 vs. 65536); this roughly halves the evaluation time and storage requirements.

Second, one observes that, typically, there is a strong degree of correlation between an approximation and the original data. This means that the values of $Q_{f,g}$ are large when f is close to g (i.e., near the diagonal of the matrix); and small, often zero, when $j \ll g$, (i.e., "far away" from the diagonal). Also, we have observed that the number of non-zero entries in a $Q_{f,g}$ table decreases for nodes closer to the leaves of the octree. This occurs because the nodes closer to the leaf nodes (a) correspond to high-resolution approximations, (b) and are a better approximation, and (c), cover a progressively smaller section of the domain; thus, the non-zero values in the $Q_{f,g}$ table cluster around the diagonal. We perform a column-major scan, i.e., traverse the table first by column and then by row, and will terminate a row early if there are no more non-zero entries on that row. We maintain a table, L_{row}, that contains the index of the last non-zero value for a row.

Third, one can use "lazy evaluation" of the error. When we re-calculate the error for all nodes in the hierarchy and few of the nodes are rendered, much of the error evaluation is wasted. Thus, for each new transfer function, we only re-calculate the error of those nodes that are being considered for rendering.

5 Results

Performance results were obtained on an SGI Origin2000 with 10GB of memory, using one (of 16) 195MHz R10K processor. We used the Visible Female CT data set, consisting of $512^2 \times 1734$

Level	Voxels	Nodes in	Nodes	MB	Rendering	Error	
		Level	Rendered		Time	l_∞	E_{rms}
0	$512^2 \times 1734$	2268	1560	390	83.4	0.00	0.00000
1	$257^2 \times 868$	350	263	65	8.73	0.3054	0.00678
2	$129^2 \times 435$	63	49	13	2.38	0.3054	0.00917
3	$65^2 \times 218$	16	16	4	0.563	0.3054	0.01059

Table 1: Visible Female CT data set multiresolution statistics. Time is in seconds.

Step	Static Texture	Index Texture
Compute Texture Hierarchy	77 min 1 sec	1 min 46 sec
Compute Error Table	-	5 min 35 sec
Calculate Error	**3 hr 27 min**	**1.23 sec**

Table 2: Performance for Visible Female CT data set. Error calculated on 432 approximating nodes.

8-bit (byte) voxels. Table 1 shows statistics for different levels (resolutions) of this data set: 1(a) shows the original data, and 1(b) to 1(d) are progressively 1/2 linear (1/8 total) size. With a tile size of 64^3, the resulting hierarchy contains 2700 nodes; 432 approximating nodes with $Q_{f,g}$ tables. The rendering performance, nodes rendered, total memory transferred to the graphics subsystem, time required to render the image, and the l_∞ and E_{rms} error values are shown for each level. Each tile contains 64^3 bytes = 256K, so the total memory transferred to the graphics subsystem for n tiles is 256K$\times n$. The "Level" column shows which level of the approximation hierarchy we are examining. The number of nodes rendered is actually less than one would expect: Many regions of the data set are constant, so there is no error when approximating these regions. Table 2 shows the advantage of our technique; even when we re-calculate the error for all nodes in the hierarchy, our method is four orders of magnitude (10000 times) faster than the method for static textures. The code for the static textures is not efficient, but there is not a 100-fold increase in efficiency left in the code. The time per node is approximately 0.0028 seconds (= 1.23 sec/432 nodes) - and when coupled with a lazy evaluation scheme, makes the error calculation almost instantaneous, or insignificant in terms of the other parts of the rendering pipeline (see "Time" column in Table 1).

A Framework for Visualizing Hierarchical Computations

Terry J. Ligocki[1], Brian Van Straalen[2], John M. Shalf[1], Gunther H. Weber[3,4], Bernd Hamann[3]

[1] Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center, 1 Cyclotron Road, M/S 50F, Berkeley, CA 94720, USA, {TJLigocki,Jshalf}@lbl.gov
[2] Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center, 1 Cyclotron Road, M/S 50A-1148, Berkeley, CA 94720, USA, BVStraalen@lbl.gov
[3] Center for Image Processing and Integrated Computing, Department of Computer Science, University of California, 1 Shields Avenue, Davis, CA 95616-8562, USA, {weber,hamann}@cs.ucdavis.edu
[4] AG Graphische Datenverarbeitung und Computergeometrie, FB Informatik, Universitaet Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany, weber@informatik.uni-kl.de

1. Introduction

Researchers doing scientific computations are attempting to accurately model physical phenomenon. When those physical phenomena take place at a variety of different scales it can be more efficient and accurate to model them at different levels of detail in an adaptive manner. Two groups here in the National Energy Research Scientific Computing center [NERSC] at the Lawrence Berkeley National Laboratory [LBNL] are doing just that. One group is headed by John Bell (the Center for Computational Sciences and Engineering, [CCSE]) and the other is headed by Phil Colella (the Applied Numerical Algorithms Group, [ANAG]). Both are doing computations using similar adaptive mesh refinement (AMR) techniques. Since the term "AMR" can mean a variety of things to researchers it should be clarified that we use it to refer exclusively to block structured AMR as defined in a paper by Berger and Colella [Berger].

Given that the researchers have already defined a hierarchical structure for their data and are performing their computations using this structure, it has been our job to provide a visualization tool that accurately represents this data. This has been a joint effort between ANAG and the LBNL/NERSC Visualization Group [Vis]. It includes extending and modifying visualization algorithms (e.g. isosurface computation, streamline generation) to correctly generate results while taking advantage of inherent computational efficiencies in the original AMR data structure. To this end we have chosen ANAG's AMR computational library [Chombo] and built an extensible visualization tool [ChomboVis] for the data sets Chombo produces. This tool was built on top of the Visualization Toolkit [VTK] using one of its interpretive interfaces [Tcl/Tk]. By using VTK we have been able to use a broad foundation of existing algorithms and infrastructure while benefiting from ongoing extensions and improvements to VTK, e.g. [Norman]. Also, because VTK is an extensible, object oriented library, we can add functionality to existing algorithms and add new algorithms relatively easily. The interpretive interface, Tcl/Tk, has allowed us to rapidly prototype ideas, add new functionality, benefit from the work other groups are doing with Tcl/Tk, and give researchers using our tool the option to directly extend it in an interactive fashion.

The development of this tool has proceeded in several directions. First, we have implemented the second version of ChomboVis, which provides the researchers with many of the tools they need to view and investigate their data. This version provides 2D and 3D visualization capabilities including that ability to look at selected data directly in a spreadsheet fashion. Second, we have been actively researching extensions of visualization techniques and algorithms to AMR data (e.g. seamless isosurface generation). These extensions can then be integrated into VTK and ChomboVis. Finally, we have been using recent extensions to VTK to handle the AMR data sets in a more natural and efficient manner. We believe that taken together, this has provided researchers with a tool that meets their immediate needs, will be extended to meet future needs, and will benefit from other work being done by the visualization research community.

2. Overview of Past and Current Work

The following is only a brief description of the AMR data generated by Chombo. It is intended to give the reader an idea of the structure of the data but not to be a precise or complete definition. The AMR data produced by computations using Chombo consists of a set of regular grids that are grouped by level. All the grids on a given level have the same cell size or resolution. All grids on a given level are completely covered by grids on the next coarser level. The ratio of the cell size on one level to the cell size on the next finer level is always an integer. Finally, the data values are all cell centered and the computations being done are finite difference approximations to partial differential equations, PDE's.

There are many approaches to extending visualization algorithms to AMR data. One of the primary difficulties is that there may be multiple data values at a given point - each values coming from a different level in the AMR hierarchy. There are several approaches that can be taken to deal with this:

1. Treat all the grids (and their values) independently.
2. Use the data value(s) from the finest grid available and ignore data value(s) from coarser grids.
3. Combine the data together in some way that is physically meaningful and use the result for visualization.

Each of these approaches can be useful depending on what the user is looking for in the data sets. If the user is debugging computational algorithms the first approach allows them to look at all the data. If the user is trying to understand and/or present computations the second or third approaches may be the best.

2.1 Visualization Tool

We started developing a visualization tool treating all grids and data independently since it was the most straightforward to implement and would be of the most immediate value to ANAG as they worked on the Chombo library. Also, it would also be of considerable value to the users of the Chombo library to have some form of visualization tool that would be extended over time. The result of this was ChomboVis. The first version was released at the beginning of 2000 and the second version was released in late 2000.

ChomboVis was built using VTK and its Tcl/Tk interface. VTK was chosen because is a freely available library that includes source code and thus can be modified and extended directly. ChomboVis contains several new VTK objects – for example, an object that reads the HDF5 output of Chombo and converts it into VTK data objects. Tcl/Tk was used to develop the user interface and to put together the VTK objects into a working system. In addition, we envision users directly interacting with ChomboVis via its Tcl/Tk interface and creating custom extensions in that manner.

The second release of ChomboVis works with 2D and 3D data sets and provides the following capabilities (see Figure 1 for an illustration of the tool in operation):

- Data selection by scalar variable and level ranges
- Orthogonal data slicing and display
- Multiple isosurface or contour generation
- Spreadsheet viewing of individual grids
- Selection of grids by pointing into the visualization
- Display of grid bounds, cell size, etc.
- Output in CGM format
- User specified colormaps

These are fairly modest capabilities and yet they required a substantial amount of development. This was because VTK (and other widely available visualization packages) do not directly support multiple grids. Much of the work we did was to provide a mechanism for sending multiple grids through a given VTK pipeline and collecting the results for rendering.

Initially we planned to use some of the recent VTK extensions to handle groups/arrays of grids [Ahrens]. This wasn't possible due to differences between the task the extensions were addressing (domain decomposition) and our task (handling overlapping sets of grids). We then turned to a novel, streaming, out-of-core technique using VTK and extensions based on work done by Matthew Hall [Hall] in order to minimize memory requirements and pipeline overhead.

2.2 Visualization Research

While the work on ChomboVis continued we began to collaborate with researchers at the Center for Image Processing and Integrated Computing [CIPIC] in the Department of Computer Science [UCD-CS] at the University of California, Davis [UCD]. This collaboration was born of our work with AMR data sets and the work at CIPIC in hierarchical representations and visualizations of large data sets. Over the summer of 2000 we began several projects with professors and graduate students from UCD. The goal of these projects was to extend visualization algorithms and techniques to AMR grids. Specifically, the following areas were investigated:

1. The generation and rendering isosurfaces from AMR data sets with no artifacts due to overlap, gaps, or cracks between grids at different levels.
2. The visualization of vector fields defined in AMR data sets.
3. The visualization of embedded boundaries, EB, which was being developed by ANAG in conjunction with their AMR work.
4. Interactive, immersive visualization of large AMR data sets using techniques that included seeded isosurface generation.
5. Interactive previewing (fast with artifacts) and high quality (slower without artifacts) volume rendering techniques for AMR data sets.

In all cases, one goal was to take advantage of the regular grid structure wherever possible and only do additional work where it was necessary. This was one of the general advantages of this type of AMR representation in scientific computations. A substantial amount of progress was made on each of these projects. The accomplishments in each area were:

1. A technique was implemented for handling multiple, overlapping grids. It removed portions of grids that overlapped finer grids, handled the rest using marching cubes [Lorensen], and then generated stitching cells and geometry at the boundaries to create a seamless result.

2. Some initial infrastructure was implemented to allow AMR vector data to be manipulated, some simple visualization of AMR vector fields was done, and a technique was implemented to compute integral curves of an AMR vector field.

3. The infrastructure to represent EB data for boundary reconstruction and visualization was implemented. Using this, several techniques for boundary reconstruction were implemented.

4. Interactive, seeded isosurface software was developed which worked with regular grids, curvilinear grids, and general tetrahedral meshes. Time budgets and work queues were used to guarantee interactivity.

5. Several promising techniques for interactive volume rendering were explored – including a technique that ignores much of the detailed structure in AMR data sets by viewing the data as a 3D point set.

Much work still needs to be done in all these areas and this will be discussed in the next section.

3. Future Work

We feel there are ample opportunities for continued work both in the form of direct extensions to current work and more subtle new directions. For example, some of the successful visualization research will be integrated into ChomboVis and use of ChomboVis will suggest new areas of visualization research.

3.1 Visualization Tool

ChomboVis is a maturing visualization tool for AMR data sets produced by Chombo. As such, it will continue to be used by researchers and improved to meet their needs. There are several aspects to this process. The user interface(s), documentation, and general usability will be improved. This can be done by working with users to get a better understanding of how they use (and abuse) ChomboVis. The performance of the tool will need to be improved for large data sets. This will require more substantial modifications to VTK to handle sets of grids in a more direct fashion. There is ongoing work in this area by other users of VTK and we plan to use this work to help us address performance issues.

Finally, we plan to integrate some of the successful AMR visualization research we being done into VTK and ChomboVis to provide users with the most advanced visualization tool we are capable of creating. Specifically, the seamless isosurface work, AMR vector visualization techniques, and interactive volume rendering work are candidates for integration.

3.2 Visualization Research

As has been said before, each area of AMR visualization research discussed contains many opportunities for continued work. Also, all the visualization techniques being developed could be studied in the interactive, immersive context being developed. Researchers will be combining AMR and EB technologies together and thus research in visualizing AMR and EB data sets will need to be combined and extended. The computing and visualizing of scalars and vectors derived from the quantities which were originally computed is of considerable interest to researchers using Chombo. Finally, applying and extending all the visualization tools and research to time varying AMR data sets provides a difficult challenge.

4. Conclusions

By working closely with researchers doing AMR computations and developing visualization tools that they use we have developed a framework for visualizing AMR computations. This has given rise to a number of visualization research questions and problems. Work on these problems has lead to extensions of our original framework and tools. This, in turn, will lead to more visualization research.

Acknowledgments

This work was supported by the Directory, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098 and by the National Science Foundation, through the Large Scientific and Software Data Set Visualization (LSSDSV) program, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research; the Army Research Office; the NASA Ames Research Center through an NRA award; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO). We also acknowledge the support of ALSTOM Schilling Robotics, Chevron, General Atomics, Silicon Graphics, Inc. and ST Microelectronics, Inc.

In addition, we would like to thank ANAG, the LBNL/NERSC Visualization Group, and the members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis for their help during the development of ChomboVis and our visualization research.

References

[Ahrens] James Ahrens, Charles Law, Will Schroeder, Ken Martin, Michael Papka, “A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets”, Los Alamos National Laboratory - Tech. Report #LAUR-00-1620.

[ANAG] http://seesar.lbl.gov/anag/

[Berger] Marsha Berger and Phil Colella, “Local adaptive mesh refinement for shock hydrodynamics”, Journal of

Computational Physics, 82:64-84, May 1989, Lawrence Livermore Laboratory Report No. UCRL-97196.

[CCSE] http://seesar.lbl.gov/ccse/

[Chombo] http://seesar.lbl.gov/anag/chombo/

[ChomboVis] http://seesar.lbl.gov/anag/chombo/chombovis.html

[CIPIC] http://graphics.cs.ucdavis.edu/

[Hall] http://zeus.ncsa.uiuc.edu/~mahall

[HDF5] http://hdf.ncsa.uiuc.edu/HDF5/

[LBNL] http://www.lbl.gov/

[Lorensen] William E. Lorensen and Harvey E. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm", Computer Graphics, 21(4):163-169, July 1987.

[NERSC] http://www.nersc.gov/

[Norman] Michael L. Norman, John Shalf, Stuart Levy, and Greg Daues, "Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations", Computing in Science and Engineering, 1(4):36-47, July/August 1999.

[Tcl/Tk] "Tcl and the Tk Toolkit", John K. Ousterhout, Addison-Wesley, 1994.

[UCD] http://www.ucdavis.edu/

[UCD-CS] http://www.cs.ucdavis.edu/

[Vis] http://www-vis.lbl.gov/

[VTK] "The Visualization Toolkit, 2nd Edition", Will Schroeder, Ken Martin, Bill Lorensen, Prentice-Hall Inc., 1998.

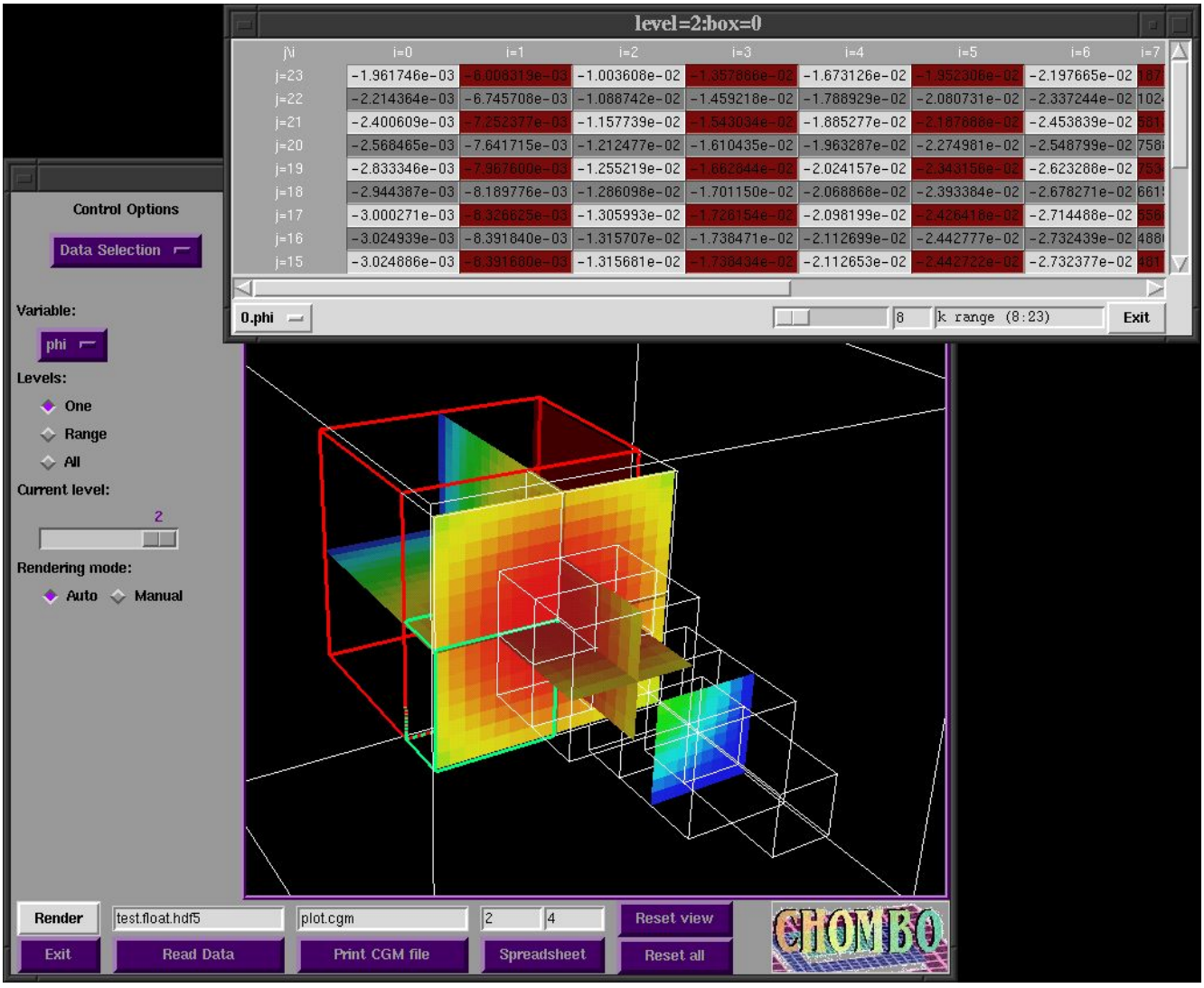


Figure 1: Visualization of an AMR data set using ChomboVis.

VR-based Rendering Techniques for Large-scale Biomedical Data Sets

Joerg Meyer
NSF Engineering Research Center (ERC) - Department of Computer Science
Mississippi State University - Box 9627 - Mississippi State, MS 39762-9627
jmeyer@cs.msstate.edu

Ragnar Borg, Bernd Hamann, Kenneth I. Joy
Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science - University of California - One Shields Ave. - Davis, CA 95616-8562
Ragnar.Borg@proxycom.no, hamann@cs.ucdavis.edu, joy@cs.ucdavis.edu

Arthur J. Olson
The Scripps Research Institute - 10550 North Torrey Pines Road - La Jolla, CA 92037
olson@scripps.edu

Abstract

VR-based rendering of large-scale data sets is typically limited by timing and complexity constraints of the rendering engine. Decentralized rendering, such as accessing a large data repository over a network and rendering the image on the client side, causes problems due to the limited bandwidth of existing networks. We present a combination of octree space subdivision and wavelet compression techniques to store large volumetric data sets in a hierarchical fashion, and we incorporate a unique numbering scheme, so that subvolumes (regions-of-interest) can be extracted efficiently at different levels of resolution.

Keywords: large-scale visualization, biomedical imaging, remote visualization, multiresolution, octree, wavelet, virtual reality

1. Introduction

We present a framework for distributed hierarchical rendering of large-scale data sets that addresses two problems at the same time: (i) limited network bandwidth and (ii) limited rendering resources. Our goal is to compactify the data set and to break it down into smaller bricks, while making effective use of multiresolution techniques. Our system uses a Windows NT-based server system which is both data repository and content provider for shared rendering applications. The client accesses the server via a web-based interface.

The client selects a data set and sends a request for information retrieval to the server. The server analyzes the request and returns a customized Java applet and the appropriate data. The Java applet is optimized for a specific rendering task. This means that the rendering algorithm is customized for a particular problem set, thus keeping the applet small by avoiding additional overhead for different cases. The initial data set is also small and will be refined later upon additional requests by the client. Our hierarchical rendering techniques include adaptive space subdivision algorithms, such as adaptive octrees for volumes, wavelet-based data reduction and storage of large volume data sets, and progressive transmission techniques for hierarchically stored volume data sets. The web-based user interface combines HTML-form-driven server requests with customized Java applets, which are transmitted by the server to accomplish a particular rendering task.

Our prototype implementation features 2-D/3-D preview capability; interactive cutting planes (in a 3-D rendering, with hierarchical isosurface models to provide context information); a lens paradigm to examine a particular region-of-interest (variable magnification and lens shape, interactively modifiable ROI); etc. Complex scenes can be precomputed on the server side and transmitted as a VRML2 file to the client so that the client can render and the user can interact with it in real time.

2. Memory-efficient storage and access

Original data are usually structured as a set of files, which represents a series of 2-D cross-sections.

Putting all those slices together, we obtain a 3-D volume. Unfortunately, when we access the data, we typically don't need the implicit coherency across single slices. This coherency stretches only across one direction. Instead, we need brick-like coherency within subvolumes. We present a new datastructure, which uses a combination of delimited octree space subdivision and wavelet compression techniques to achieve better performance.

We present an efficient indexing scheme, a suitable data reduction method, and an efficient compression scheme. All techniques are based on integer arithmetic and are optimized for speed. Binary bit operations allow for memory efficient storage and access.

We use the standard filesystem to store our derived datastructures, and we use filenames as keys to the database, thus avoiding additional overhead, which is typically caused by adding additional layers between the application and the underlying storage system. We found that this method provides the fastest method to access the data. Our indexing scheme in conjunction with the underlying filesystem provides the database system (repository) for the server application, which reads the data at a low resolution from the repository and sends it to a remote rendering client upon request. After the user has specified a subvolume or region-of-interest (ROI), the client application sends a new request to the server to retrieve a subvolume at a higher level of resolution. This updating procedure typically takes considerably less time, because only a small number of files need to be touched. The initial step, which requires to read the initial section of every file, i.e., all bricks, can be sped up by storing an additional file which contains a reduced version of the entire data set.

Our new data structure uses considerably less memory than the original data set, even if the user chooses lossless compression (see statistics, chapter 6). By choosing appropriate thresholds for wavelet compression, the user can switch between lossless compression and extremely high compression rates. Computing time is balanced by choosing an appropriate filesize (chapter 3).

One of the advantages of this approach is the fact that the computing time does not so much depend on the resolution of the subvolume, but merely on the size of the subvolume. This is because the higher resolution versions (detail coefficients in conjunction with the lower resolution versions) can be retrieved in almost the same time from disk as the lower resolution version alone. All levels of detail are stored in the same file, and the content of several files, which make up the subvolume, usually fits into main memory. Since seek time is much higher than read time for conventional harddisks, the total time for data retrieval mainly depends on the size of the subvolume, i.e., the number of files that need to be accessed, and not so much on the level of detail.

3. Filesize considerations

The filesize f for storing the leaves of the octree structure, which is described in chapter 4, should be a multiple n of the minimum page size p of the filesystem. p is typically defined as a system constant in `/usr/include/sys/param.h`). n depends on the wavelet compression. If the lowest resolution of the subvolume requires b bytes, the next level requires a total of $8 \cdot b$ bytes (worst case, uncompressed) and so forth.

We assume that we have a recursion depth r for the wavelet representation. This gives us $8^r \cdot b$ bytes, which must fit in f. This means:

$$f = n \cdot p \geq 8^r \cdot b$$

Both r and b are user-defined constants. Typical values are $b = 512$, which corresponds to an $8 \times 8 \times 8$ subvolume, and $r = 3$, which gives us four levels of detail over a range between 512 and $8^3 \cdot 512 = 262144$ data elements, which is more than 2.7 orders of magnitude.

For optimal performance and in order to avoid gaps in the allocated files, we can assume that

$$n \cdot p = 8^r \cdot b,$$

thus

$$n = 8^r \cdot \frac{b}{p}.$$

4. Delimited octree and wavelet structure

The enormous size of the data sets (see chapter 5) requires to subdivide the data into smaller chunks, which can be loaded into core memory within a reasonable amount of time [Hei98, Mey97]. Since we are extracting subvolumes, it seems quite natu-

ral to break the data up into smaller bricks. This can be done recursively by using an octree approach [Jac80, Mea80, Red78]. Each octant is subdivided until we reach an empty region which does not need to be subdivided any further, or until we hit the filesize limit *f*, which means that the current leaf fits into a file of the given size.

Each leaf contains the full resolution. The memory reduction occurs by skipping the empty regions. Typically, the size of the data set shrinks to about 20%, i.e., one fifth of the original size (see chapter 6).

Since we want to access the data set in a hierarchical fashion, we have to convert the leaves into a multiresolution representation. This representation must be chosen in a way that the reconstruction can be performed most efficiently with minimal computational effort. Haar wavelets fulfill these properties. They also have the advantage that they can be easily implemented as integer arithmetic. The lower resolution is stored at the beginning of the file, thus avoiding long seek times within the file.

Another very useful property is the fact that a volume converted into the frequency domain, i.e., the wavelet representation, requires the exact same amount of memory as the original representation. This is also true for all subsequent wavelet recursions.The wavelet recursion terminates when we have reached a predifined minimum subvolume size *b*. The lower bound is the size of a single voxel.

Each octant can be described by a number [Fol96, Hun79]. We use the following numbering scheme (figure 1): A leaf is uniquely characterized by the octree recusion depth and the octree path. We limit the recursion depth to eight, which allows us to encode the depth in 3 bits. In order to store the path, we need 3 bits per recursion step, which gives us 24 bits. 4 bits are spent to encode the depth of the wavelet recursion. The remaining bit is a flag which indicates that the file is empty. This prevents us from opening and attempting to read the file and speeds up the computation. The total number of bits is 32 (double word).

3	3		3	4	1
oct.depth	sub 1	. . .	sub 8	wav.depth	empty

Each bit group can be easily converted into an ASCII character by using binary arithmetic, e.g., `(OCT_DEPTH >> 29) | 0x30)` would encode the octree depth as an ASCII digit. By appending these characters we can generate a unique filename for each leaf.

In order to retrieve a subvolume, we have to find the file(s) in which it is stored. We start with the lower left front corner and identify the subvoxel by recursive binary subdivision of the bounding box for each direction. Each decision gives us one bit of the subvolume path information. We convert these bits into ASCII characters, using the same macros as above. The first file we are looking for is `7xxxxxxxx??`, where the 'x's describe the path, and '?' is a wildcard. If this file does not exist, we keep looking for `6xxxxxxx???`, and so forth, until we find an existing leaf. If the filename indicates that the file is empty (last digit), we can skip the file. The filename also indicates how many levels of detail we have available for a particular leaf. This allows us to scale the rendering algorithm. In order to retrieve the rest of the subvolume, we must repeat this procedure for the neighboring leaves. The number of iterations depends on the recusion depth and therefore on the size of the leaves found. The algorithm terminates when all files have been retrieved so that the subvolume is complete.

5. Applications

Our test applications include molecular biology, medicine, and earthquake simulation. Our prototype for the biomedical field was designed to support three-dimensional visualization of a human brain, which allows us to study details by moving tools, such as an arbitrary cutting plane and variously shaped lenses, across the data set. The various data sets are typically between 20 MB and 76 GB, which makes them impossible to transfer over the internet in real time. The rendering client operates independently from the size of the data set and requests only as much data as can be displayed and handled by the Java applet.

6. Statistics

Table 1 shows the reduction of memory which is required to store a large data set, if we use an octree at two different levels. The column on the right represents the original data set. The wavelet decomposition takes about 0.07 sec for a 64^3 data set, and 68 sec for a 1024^3 data set. The recon-

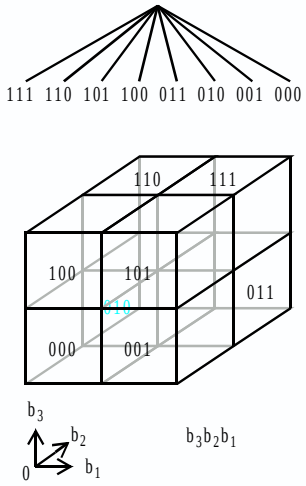


Fig. 1: Numbering scheme

Algorithm	Octree			
	level 1		level 2	
Data type	MRI	CT	MRI	CT
Pre-processing	56	63	98	97
Depth	4	4	5	5
Memory	5.412.610 / 14.548.992	3.996.526 / 14.811.136	3.831.488 / 14.548.992	2.358.442 / 14.811.136

Tab. 1: Space subdivision algorithm

struction can be done more efficiently and usually takes about 30% of the time (measurements based on an R12000 processor). For the above data we assume lossless wavelet decomposition. RLE or other (lossy) compression/decompression algorithms take an additional amount of time and will be implemented in a later version. We will choose algorithms with asymmetric behavior, i.e., compression time is higher than decompression time.

7. Conclusions

We have presented an efficient numbering scheme and access method for hierarchical storage of subvolumes on a regular filesystem. This method allows us to access a region-of-interest as a set of bricks at various resolutions. The simplicity of the method makes it easy to implement. The algorithm easily scales by increasing word length and filename length. Future work includes better wavelet compression schemes and time-variant data sets.

We are currently working on the integration, adaptation and evaluation of these tools in the National Partnership for Advanced Computational Infrastructure (NPACI) framework. Integration of San Diego Supercomputer Center's High-performance Storage System (HPSS) as a data repository to retrieve large-scale data sets, accessing the data via NPACI's Scalable Visualization Toolkits (also known as VisTools), and evaluation of particular applets with NPACI partners, are main goals for future research efforts.

Acknowledgements

NSF (CAREER Awards: ACI 9624034, ACI 9983641, LSSDSV: ACI 9982251, NPACI); Office of Naval Research (ONR, N00014-97-1-0222); Army Research Office (ARO 36598-MA-RIP); NASA Ames Research Center (NAG2-1216); LLNL (ASCI ASAP Level-2, B347878, B503159); NATO (CRG.971628); ALSTOM Schilling Robotics; Chevron; Silicon Graphics, Inc.; ST Microelectronics, Inc.; Data sets courtesy of Arthur W. Toga, UCLA School of Medicine, Arthur J. Olson, The Scripps Institute, and Edward G. Jones, Neuroscience Center, UC Davis.

References

[Hei98] Heiming, Carsten, “Raumunterteilung von Volumendaten,” *thesis*, Department of Computer Science, University of Kaiserslautern, Germany, January 1998.

[Hun79] Hunter, G. M.; Steiglitz, K., “Operations on Images Using Quad Trees,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2), April 1979, 145–154.

[Jac80] Jackins, C.; Tanimoto, S. L., “Oct-Trees and Their Use in Representing Three-Dimensional Objects,” *CGIP*, 14(3), November 1980, 249–270.

[Mea80] Meagher, D., “Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer,” *Technical Report IPL-TR-80-111*, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, October 1980.

[Mey97] Meyer, Jörg; Gelder, Steffen; Heiming, Carsten; Hagen, Hans, “Interactive Rendering—A Time-Based Approach,” *SIAM Conference on Geometric Design ’97*, Nashville, TN, November 3–6, 1997, 23.

[Red78] Reddy, D.; Rubin, S., “Representation of Three-Dimensional Objects,” *CMU-CS-78-113*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

[Sch97] Schneider, Timna Esther, “Multiresolution-Darstellung von 2D-Schichtdaten in der medizinischen Bildverarbeitung,” *thesis*; Department of Computer Science, University of Kaiserslautern, Germany, December 1997.

Multi-resolution Indexing for Hierarchical Out-of-core Traversal of Rectilinear Grids

Valerio Pascucci

1 Introduction

The real time processing of very large volumetric meshes introduces specific algorithmic challenges due to the impossibility of fitting the input data in the main memory of a computer. The basic assumption (RAM computational model) of uniform-constant-time access to each memory location is not valid because part of the data is stored out-of-core or in external memory. The performance of most algorithms does not scale well in the transition from the in-core to the out-of-core processing conditions. The performance degradation is due to the high frequency of I/O operations that may start dominating the overall running time.

Out-of-core computing [22] addresses specifically the issues of algorithm redesign and data layout restructuring to enable data access patterns with minimal performance degradation in out-of-core processing. Results in this area are also valuable in parallel and distributed computing where one has to deal with the similar issue of balancing processing time with data migration time.

The solution of the out-of-core processing problem is typically divided into two parts:

(i) algorithm analysis to understand its data access patterns and, when possible, redesign to maximize their locality;

(ii) storage of the data in secondary memory with a layout consistent with the access patterns observed to amortize the cost of each I/O operation over several memory access operations.

In the case of a hierarchical visualization algorithms for volumetric data the 3D input hierarchy is traversed to build derived geometric models with adaptive levels of detail. The shape of the output models is then modified dynamically with incremental updates of their level of detail. The parameters that govern this continuous modification of the output geometry are dependent on the runtime user interaction making it impossible to determine a priori what levels of detail are going to be constructed. For example they can be dependent on external parameters like the viewpoint of the current display window or on internal parameters like the isovalue of an isocontour or the position of an orthogonal slice. The structure of the access pattern can be summarized into two main points: (i) the input hierarchy is traversed level by level so that the data in the same level of resolution or in adjacent levels is traversed at the same time and (ii) within each level of resolution the data is mostly traversed at the same time in regions that are geometrically close.

In this paper I introduce a new static indexing scheme that induces a data layout satisfying both requirements (i) and (ii) for the hierarchical traversal of n-dimensional regular grids. In one particular implementation the scheme exploits in a new way the recursive construction of the Z-order space filling curve. The standard indexing that maps the input nD data onto a 1D sequence for the Z-order curve is based on a simple bit interleaving operation that merges the n input indices into one index n times longer. This helps in grouping the data for geometric proximity but only for a specific level of detail. In this paper I show how this indexing can be transformed into an alternative index that allows to group the data per level of resolution first and then the data within each level per geometric proximity. This yields a data layout that is appropriate for hierarchical out-of-core processing of large grids.

The scheme has three key features that make it particularly attractive. First the order of the data is independent of the out-of-core blocking factor so that its use in different settings (e.g. local disk access or transmission through a network) does not require large data reorganization. Secondly the conversion from the standard Z-order indexing to the new index can be implemented with a simple sequence of shift operations making it appealing for a possible hardware implementation. Third there is no data replication which is especially desirable when the data is accessed through slow connections and avoids eventual problems of data consistency and multiple I/O operations when the data ismodified.

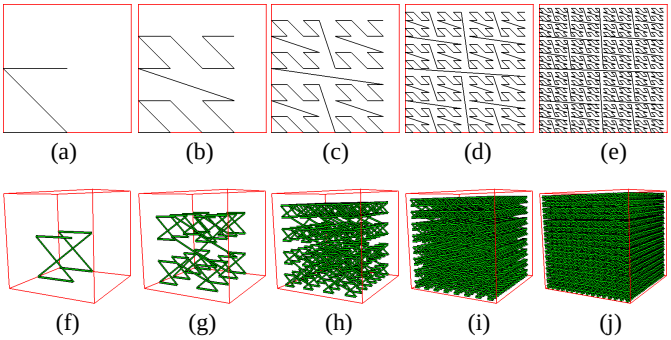


Figure 1: (a-e) The first five levels of resolution of the 2D Lebesgue's space filling curve. (f-j) The first five levels of resolution of the 3D Lebesgue's space filling curve.

Beyond the theoretical interest in developing hierarchical indexing schemes for n-dimensional space filling curves the approach is currently targeted for its practical use in out-of-core visualization algorithms. Experimental results and theoretical analysis are reported in this paper for the simple case of orthogonal slicing of volumetric data. The results show how the practical performance enhancement corresponds to the theoretical expectations. The scheme is also targeted to perform out-of-core progressive computation of general slices, for its use in combination with the 3D progressive isocontouring algorithm [18] and for out-of-core visualization of large terrains using edge bisection hierarchies [8, 13]. Extensions are being considered for larger classes of datasets and the combined use with wavelet representation.

2 Related Previous Work

External memory algorithms [22], also known as out-of-core algorithms, have been rising in recent years to the attention of the computer science community since they address systematically the problem of non uniform memory structure of modern computers (fast cache, main memory, hard disk, ...). This issue is particularly important when dealing with large data-structures that do not fit in the main memory of a single computer since the access time to each memory unit is dependent on its location. New algorithmic techniques and analysis tools have been developed to address this problem for example in the case of geometric algorithms [14, 10, 1] or scientific visualization [7, 3]. Closely related issues emerge in

the area of parallel and distributed computing where remote data transfer can become a primary bottleneck in the computation. In this context space filling curves [20] are often used as a tool to determine very quickly data distribution layouts that guarantee good geometric locality [17, 11, 15]. Space filling curves have been also used in the past in a wide variety of applications [2] both because of their hierarchical fractal structure as well as for their well known spatial locality properties. The most popular is the Hilbert curve which guarantees the best geometric locality properties [16]. The pseudo-Hilbert scanning order [5] generalizes the scheme to rectilinear grids that have different number of samples along each coordinate axis.

Recently Lawder [12] explored the use of different kinds of space filling curves to develop indexing schemes for data storage layout and fast retrieval in multi-dimensional databases.

Balmelli at al. [4] use the Z-order space filling curve to navigate efficiently a quad-tree data-structure without using pointers. They use simple closed formulas for computing neighboring relations and nearest common ancestors between nodes to allow fast generation of adaptive edge-bisection triangulations. They improve on the basic data-structure already used for terrain visualization [8, 13] or adaptive mesh refinement [19]. The use of the Z-order space filling curve for traversal of quadtrees [21] (also called Morton-order) has been also proven useful in the speedup of matrix operations allowing to make better use of the memory cache hierarchies [6, 23, 9].

In the approach proposed here a new data layout is used to allow efficient progressive access to volumetric information stored in external memory. This is achieved by combining interleaved storage of the levels in the data hierarchy while maintaining geometric proximity within each level of resolution. One main advantage is that the resulting data layout is independent of the particular adaptive traversal of the data. This improves fundamentally from the previous schemes since they used the space filling curves only for computation and dynamic relocation of data layouts for single resolution or fixed adaptive resolution meshes.

3 The General Framework

Consider a set S of n elements decomposed into a hierarchy $\mathcal{H}$ of k levels of resolution $\mathcal{H} = \{S_0, S_1, \ldots, S_{k-1}\}$ such that:

$$S_0 \subset S_1 \subset \cdots \subset S_{k-1} = S$$

where S_i is said to be coarser than S_j iff $i < j$. The order of the elements in S is defined by the cardinality function $I : S \to \{0 \ldots n-1\}$. This means that the following identity always holds:

$$S[I(s)] \equiv s$$

where the square brackets are used to index an element in a set.

Let's define a derived sequence $\mathcal{H}'$ of sets S'_i as follow:

$$S'_i = S_i \backslash S_{i-1} \qquad i = 0, \ldots, k-1$$

where formally $S_{-1} = \emptyset$. The sequence $\mathcal{H}' = \{S'_0, S'_1, \ldots, S'_{k-1}\}$ is a partitioning of S. A derived cardinality function $I' : S \to \{0 \ldots n-1\}$ can be defined on the basis of the following two properties:

- $\forall s, t \in S'_i : I(s) < I(t) \Rightarrow I'(s) < I'(t)$;
- $\forall s \in S'_i, \forall t \in S'_j : i < j \Rightarrow I'(s) < I'(t)$.

If the original function I has strong locality properties when restricted to any level of resolution S_i then the cardinality function I' generates the desired global index for hierarchical and out-of-core traversal.

The construction of the function can be achieved in the following way: (i) determine the number of elements in each derived set S'_i and (ii) determine a local cardinality function $I''_i = I'|_{S'_i}$ restriction of I' to each set S'_i. In particular if c_i is the number of elements of S'_i one can predetermine the starting index of the elements in a given level of resolution by building the sequence of constants $C_0, \ldots, C_{k-1}$ with

$$C_i = \sum_{j=0}^{i-1} c_j. \tag{1}$$

Secondly one needs to determine a set of local cardinality functions $I''_i : S'_i \to \{0 \ldots c_i - 1\}$ so that:

$$\forall s \in S'_i : I'(s) = C_i + I''_i(s). \tag{2}$$

The computation of the constants C_i can be performed in a preprocessing stage so that the computation of I' is reduced to the following two steps:

- given s determine its level of resolution i (that is the i such that $s \in S'_i$);
- compute $I''_i(s)$ and add it to C_i.

These two steps need to be performed very efficiently because they are going to be executed repeatedly at run time. The following sections report practical realizations of this scheme for rectilinear cube grids in any dimension.

4 Binary Tree Hierarchy

This section reports the details on how to derive from the Z-order space filling curve the local cardinality functions I''_i for a binary tree hierarchy in any dimension.

4.1 Indexing the Lebesgue Space Filling Curve

The Lebesgue space filling curve, also called Z-order space filling curve for its shape in the 2D case, is depicted in figure 1. In the 2D case the curve can be defined inductively by a base Z shape of size 1 (figure 1a) whose vertices are replaced each by a Z shape of size $\frac{1}{2}$. The vertices obtained are then replaced by Z shapes of size $\frac{1}{4}$ (figure 1c) and so on. In general the i^{th} level of resolution is defined as the curve obtained by replacing the vertices of the $(i-1)^{th}$ level of resolution with Z shapes of size $\frac{1}{2^i}$. The 3D version of this space filling curve has the same hierarchical structure with the only difference that the basic Z shape is replaced by a connected pair of Z shapes lying on the opposite faces of a cube as shown in Figure 1f. Figure 1f-j shows five successive refinements of the 3D Lebesgue space filling curve. The d-dimensional version of the space filling curve has also the same hierarchical structure where the basic shape (the Z of the 2D case) is defined as a connected pair of $(d-1)$-dimensional basic shapes lying on the opposite faces of a d-dimensional cube.

The property that makes the Lebesgue's space filling curve particularly attractive is the easy conversion from the d indices of a d-dimensional matrix to the 1D index along the curve. If one element e has d-dimensional reference $(i_1, \ldots, i_d)$ its 1D reference is built by interleaving the bits of the binary representations of the indices $i_1, \ldots, i_n$. In particular if i_j is represented by the string of h bits "$b_j^1 b_j^2 \cdots b_j^h$" (with $j = 1, \ldots, d$) then the 1D reference of e is represented the string of hd bits $I =$ "$b_1^1 b_2^1 \cdots b_d^1 b_1^2 b_2^2 \cdots b_d^2 \cdots b_1^h b_2^h \cdots b_d^h$". Figure 2 shows this interleaving scheme in the 3D case.

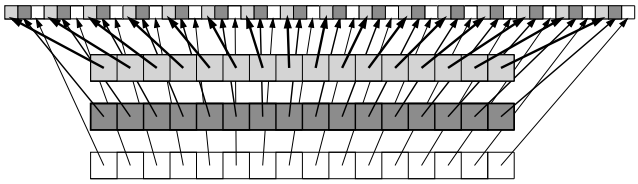


Figure 2: Construction of the 1D index from the Lebesgue's Z-order space filling curve. In the 3D case the original index is a set of three bit-strings. The 1D index is formed by interleaving the bit of the three sequences into a single bit-string.

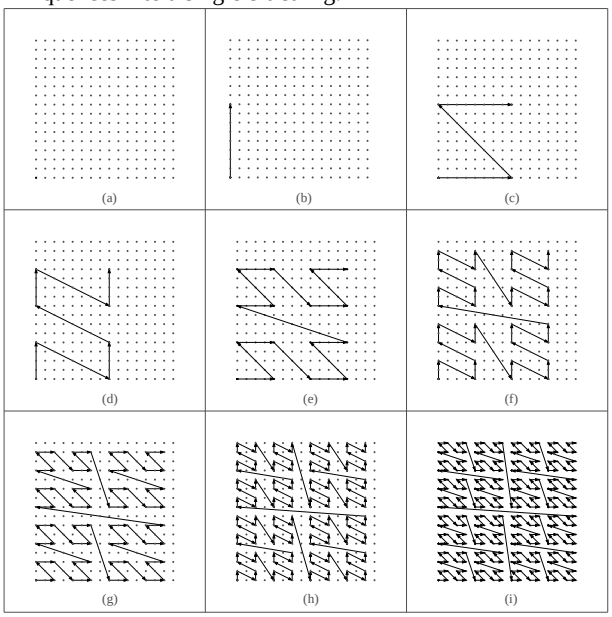


Figure 3: The nine levels of resolution of the binary tree hierarchy defined by the 2D space filling curve applied on a 16×16 rectilinear grid. The coarsest level of resolution (a) is a single point. The number of points that belong to the curve at any level of resolution (b-i) is double the number of points of the previous level.

The 1D order can be structured in a binary tree by considering elements of level i those that have the last i bits all equal to 0. This yields a hierarchy where each level of resolution has twice as many points as the previous level. From a geometric point of view this means that the density of the points in the d-dimensional grid is doubled alternatively along each coordinate axis. Figure 3 shows the binary hierarchy in the 2D case where the resolution of the space filing curve is doubled alternatively along the x and y axis. The coarsest level (a) is a single point, the second level (b) has two points , the third level (c) has four points (forming the Z shape) and so on.

4.2 Index Remapping

The cardinality function discussed in section 3 for a binary tree case has the structure shown in table 1. Note that this is a general structure suitable for out-of-core storage of static binary trees. It is independent of the dimension d of the grid of points or of the Z-order space filling curve.

The structure of the binary tree defined on the Z-order space filling curve allows to determine easily the three elements that are necessary for the computation of the cardinality which are: (i) the level i of an element, (ii) the constants C_i of equation (1) and (iii) the local indices I_i''.

i - if the binary hierarchy has k levels then the element of Z-order index j belongs to the level $k - h$ where h is the number of trailing zeros in the binary representation of j;

C_i - the number of elements in the levels coarser than i is $C_i = 2^{i-1}$ for $i > 0$, with $C_0 = 0$;

I_i'' - if an element has index j and belongs to the set S_i' then $\frac{j}{2^i}$ is an odd number. Its local index is then:

$$I_i''(j) = \left\lfloor \frac{j}{2^{i+1}} \right\rfloor .$$

These three elements can be put together to build an efficient algorithm that computes the hierarchical index $I'(s) = C_i + I_i''(s)$ in the two steps shown in the diagram of Figure 4:

1. set to 1 the bit in position $k + 1$;
2. shift to the right until a 1 comes out of the bit-string.

Clearly this diagram could have a very simple and efficient hardware implementation. The software C++ version can be implemented as follows:

```
inline adhocidex remap(register adhocindex i){
  i |= last_bit_mask;  // set leftmost one
  i /= i&-i;           // remove trailing zeros
  return (i>>1);       // remove rightmost one
}
```

This code would work only on machines with two's complement representation of numbers. In a more portable version one needs to replace `i /= i&-i` with `i /= i&((~i)+1)`.

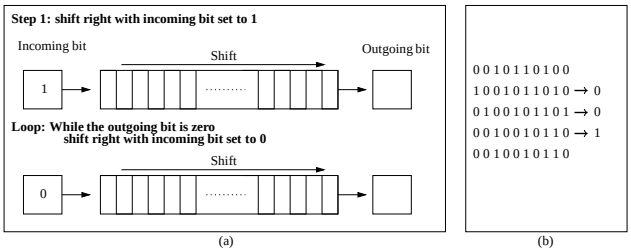


Figure 4: (a) Diagram of the algorithm for index remapping from Z-order to the hierarchical out-of-core binary tree order. (b) Example of the sequence of shift operations necessary to remap an index. The top element is the original index the bottom is the output remapped index.

5 2^l-tree Hierarchy

In some occurrences it might be appropriate to use a blocking scheme that follows better the underlying quad-tree/octtree structure of the hierarchy. For example when large disk blocks are being used it is better to use a hierarchy with less levels of resolution than the basic binary tree scheme of the previous section. At this end one can apply the following generalized version of the index remapping scheme.

The scheme of Figure 4 correspond to a binary tree because the sequence of shifts in the loop are performed one bit at a time. To align the data layout to a 2^l tree one needs to shift the bit-string by l bits at each step. For example $l = 2$ aligns the data to a quad-tree and $l = 3$ aligns the data to an octtree. Following the notation of section 3 we have:

level	0	1	2		3				4							
Z-order index (2 levels)	0	1														
Z-order index (3 levels)	0	2	1	3												
Z-order index (4 levels)	0	4	2	6	1	3	5	7								
Z-order index (5 levels)	0	8	4	12	2	6	10	14	1	3	5	7	9	11	13	15
hierarchical index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 1: Structure of the hierarchical indexing scheme for binary tree combined with the order defined by the Lebesgue space filling curve.

i - is the number of trailing groups of l zeros at the end of the bit-string;

C_i - the number of elements in the levels coarser than i is $C_i = \left(2^l\right)^{i-1}$ for $i > 0$, with $C_0 = 0$;

I_i'' - if an element has index j and belongs to the set S_i' then $\left\lfloor \frac{j}{2^i} \right\rfloor$ has one of its last l bits different from 0. The local index is then:

$$I_i''(j) = \left\lfloor \frac{j}{2^{li}} \right\rfloor - \left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor - 1.$$

The computation of the final index $I'(s) = C_i + I_i''(s)$ is then performed with the scheme of Figure 5. Note how the term sum $C_i + I_i''(s)$ is computed directly by shifting $I(s)$ to the right and adding the complement of its high bits. This can be done because $C_i = 2^{l(i-1)}$ and hence $C_i - \left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor - 1$ can be obtained directly by complementing the rightmost $l(k-i-1)$ bits of $\left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor$.

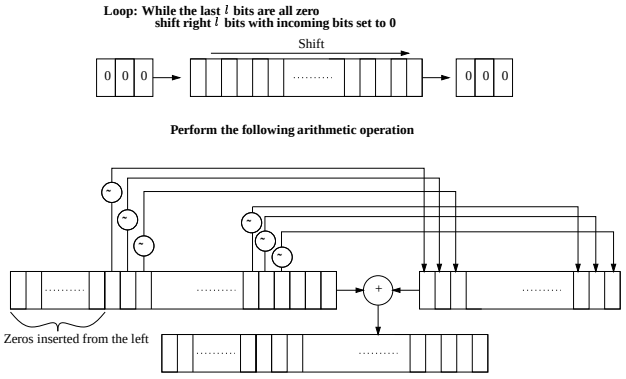


Figure 5: Diagram of the algorithm for index remapping for blocks that scale with a factor of 8 (three bits).

6 Preliminary Tests: Orthogonal Slicing

This section presents some preliminary results based on the basic and widely used application of computing orthogonal slices of a 3D rectilinear grid of data at different levels of resolution. The data layout proposed here is compared with the two most common array layouts: the standard row major structure and the $h \times h \times h$ block-wise decomposition of the data. Both practical performance tests and formal complexity analysis lead to the conclusion that the data layout proposed here allows to achieves substantial speedup when used at coarse resolution or in a progressive fashion with acceptable performance penalty if used only at the highest level of resolution.

6.1 Out-of-core Complexity Analysis

The out-of-core analysis reports the number of data blocks transferred from disk under the assumption that each block of data of size b is transferred in one operation independently of how much data in the block is actually used. At fine resolution the simple row major array storage achieves the best and worst performances depending on the slicing direction. If the overall grid size is g and the size of the output is t then the best slicing direction requires to load $O(t/b)$ data blocks (which is optimal) but the worst possible direction requires to load $O(t)$ blocks (for $b = \Omega(\sqrt[3]{g})$). In the case of simple $h \times h \times h$ data blocking (which has best performance for $h = \sqrt[3]{b}$) the blocks of data loaded at fine resolution are $O(\frac{t}{\sqrt[3]{b^2}})$. Note that this is much better than the previous case because the performance is close to (even if not) optimal independently of the particular slicing direction. For subsampling rate of k the performance degrades to $O(\frac{tk^2}{\sqrt[3]{b^2}})$ for $k < \sqrt[3]{b}$. This means that at coarse subsampling the performance goes down to $O(t)$. The advantage of the scheme proposed here is that independently of the level of subsampling each block of data is used for a portion of $\sqrt[3]{b}^2$ so that independently of the slicing direction and subsampling rate the worst case performance is $O(\frac{t}{\sqrt[3]{b^2}})$. This implies that the fine resolution performance of the scheme is equivalent to the standard blocking scheme while at coarse resolutions it can get orders of magnitude better. More importantly this allows to produce coarse resolution outputs at interactive rate independently of the total size of the data-set.

6.2 Experimental Tests

A series of tests have been performed to verify the performance of the approach. The out-of-core component of the scheme has been implemented simply by mapping a 1D array of data to a file on disk using the `mmap` function. In this way the I/O layer is implemented by the operating system that pages in and out a portion of the data array as needed. No multi-threaded component is used to avoid blocking the application while retrieving the data. The blocks of data defined by the system are typically 4Kbytes. Figure 6(a) shows performance tests executed on a Pentium III Laptop 500Mhz with 128M of RAM accessing a 1Gbyte dataset ($1k \times 1k \times 1k$ grid of char). The two schemes proposed here, 3-bits shift from section 5 and 1-bits shift from section 4 show the best scalability in performance. The blocking scheme with 16^3 chunks of regular grid shows the next best compromise in performance. The (i, j, k) row major storage scheme has the worst performance compromise because of its dependency on the slicing direction: best for (j, k) plane slices and worst for (j, i) plane slices. Figure 6(b) shows the performance results for a test run on an SGI MIPS R12000 300Mhz with 600M of memory available for the application. In this case the dataset is 8G ($2k \times 2k \times 2k$ grid of char).

7 Conclusions and Future Direction

The present paper introduces a new indexing and data layout scheme that is useful for out-of-core hierarchical traversal of large datasets. Practical tests and theoretical analysis for a simple case of orthogonal slicing show the performance improvements that can be achieved with this approach especially in a progressive computation setting. This scheme is also going to be used as a basis for

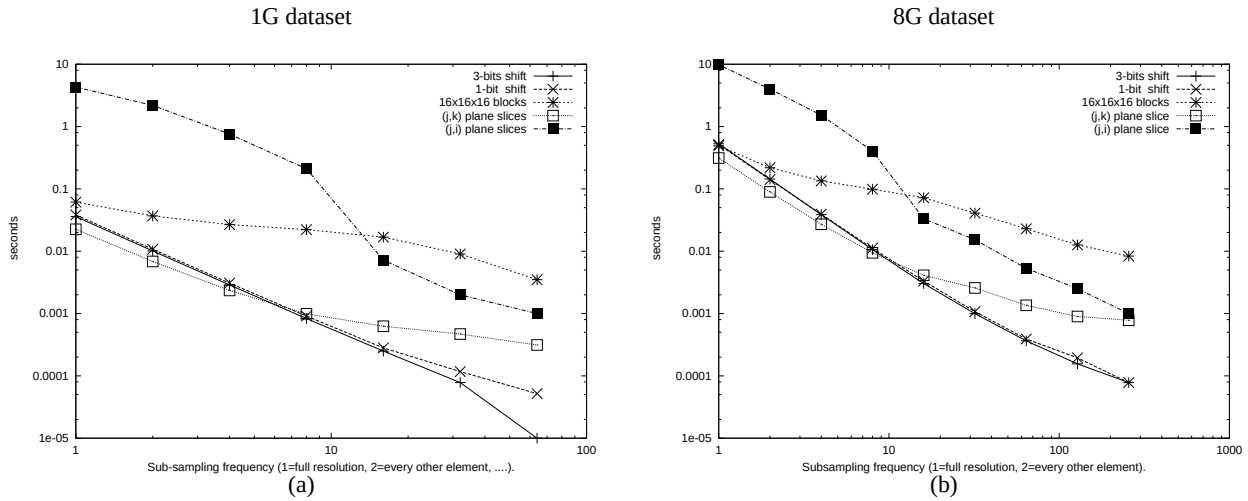


Figure 6: Two comparisons of the computation times of four different data layout schemes. The vertical axis is the computation time in seconds. The horizontal axis is the level of subsampling of the slicing scheme (test at the finest resolution are on the left). The two schemes proposed here, 3-bits shift from section 5 and 1-bits shift from section 4, show best overall performance.

out-of-core computation of general slices, progressively computed isosurfaces and navigation of large terrains.

Future direction that are being considered include the combination with wavelet compression schemes, the extension to general rectangular grids and to non-rectilinear hierarchies.

References

[1] James Abello and Jeffrey Scott Vitter, editors. *External Memory Algorithms and Visualization*.

[2] T. Asano, D. Ranjan, T. Roos, and E. Welzl. Space filling curves and their use in the design of geometric data structures. *Lecture Notes in Computer Science*, 911:36–44, 1995.

[3] C. L. Bajaj, V. Pascucci, D. Thompson, and X. Y. Zhang. Parallel accelerated isocontouring for out-of-core visualization. In Stephan N. Spencer, editor, *Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium (PVGS'99)*, pages 97–104, N.Y., October 25–26 1999. ACM Siggraph.

[4] L. Balmelli, J. Kovačević, and M. Vetterli. Quadtree for embedded surface visualization: Constraints and efficient data structures. In *IEEE International Conference on Image Processing (ICIP)*, Kobe Japan, October 1999.

[5] Y. Bandou and S.-I. Kamata. An address generator for an n-dimensional pseudo-hilbert scan in a hyper-rectangular parallelepiped region. In *International Conference on Image Processing, ICIP 2000*, 2000. to appear.

[6] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, Saint-Malo, France, June 27–30, 1999. SIGACT/SIGARCH and EATCS.

[7] Yi-Jen Chiang and Cláudio T. Silva. I/O optimal isosurface extraction. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization ý7*, pages 293–300. IEEE, November 1997.

[8] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *IEEE Visualization '97*, pages 81–88.

[9] Jeremy D. Frens and David S. Wise. Auto-blocking matrix-multiplication or tracking BLAS3 performance from source code. *ACM SIGPLAN Notices*, 32(7):206–216, July 1997.

[10] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, Palo Alto, CA, November 1993.

[11] M. Griebel and G. W. Zumbusch. Parallel multigrid in an adaptive pde solver based on hashing and space-filling curves. 25:827:843, 1999.

[12] J. K. Lawder. *The Application of Space-filling Curves to the Storage and Retrieval of Multi-Dimensional Data.* PhD thesis, School of Computer Science and Information Systems, Birkbeck College, University of London, 2000.

[13] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gregory Turner. Real-time, continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH 96*, pages 109–118, August 1996.

[14] Y. Matias, E. Segal, and J. S. Vitter. Efficient bundle sorting. In *Proceedings of the 11th Annual SIAM/ACM Symposium on Discrete Algorithms* , 2000.

[15] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in meshindexings, 1997.

[16] Rolf Niedermeier and Peter Sanders. On the manhattan-distance between points on space-filling mesh-indexings. Technical Report iratr-1996-18, Universität Karlsruhe, Informatik für Ingenieure und Naturwissenschaftler, 1996.

[17] M. Parashar, J.C. Browne, C. Edwards, and K. Klimkowski. A common data management infrastructure for adaptive algorithms for pde solutions. In *SuperComputing 97*, 1997.

[18] V. Pascucci and C. L. Bajaj. Time critical isosurface refinement and smoothing. In *IEEE Symposium on Volume Visualization and Graphics 2000*, October 2000. To Appear.

[19] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.

[20] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, New York, NY, 1994.

[21] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, Mass., 1990.

[22] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 2000.

[23] David S. Wise. Ahnentafel indexing into morton-ordered arrays, or matrix locality for free. In *Euro-Par 2000 – Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 774–784. Springer, August 2000.

A Data Model for Multiresolution Scientific Data Environments

Philip J. Rhodes, R. Daniel Bergeron, and Ted M. Sparr
Computer Science Department
University of New Hampshire
Durham, NH, 03824
{rhodes,rdb,tms}@cs.unh.edu

1. Introduction

Modern dataset sizes present major obstacles to understanding and interpreting the significant underlying phenomena represented in the data. There is a critical need to support scientists in the process of interactive exploration of these very large data sets. The goal of this exploration is to use coarse representations of broad views of the data set for purposes of identifying potentially interesting regions followed by narrower views at higher resolutions.

1.1 Problem Definition

The research described here focuses on developing database support for this method of scientific research based on interactive exploration of very large distributed multiresolution data. We believe that a major weakness of current scientific database efforts is the lack of a comprehensive model that encapsulates the structure inherent in the data. Such a model should allow a database system to store and access this data efficiently without needing to understand the meaning of the data for the application domain. The most important requirements for a data model for distributed multiresolution data include the following:

1. The model must be general-purpose while still able to rigorously encapsulate the most important aspects of the data.
2. In addition to describing the multiple resolution levels of a data set, it must be able to describe an *adaptive resolution* level, i.e., a single level of the data set that is itself composed of data that has different resolutions in different regions.
3. It must be able to describe both the physical domain in which the scientific phenomenon actually occurs (we call this the *geometry* of the problem) as well as the structure of the data that represents that phenomenon (which we call the *topology* of the data).
4. It must incorporate information about how data points in each level of the multiresolution representation relate to data points in the other levels. We have developed notions of *support* and *influence* which encapsulate these relations in an application-independent manner.

1.2 Summary of Research

The major components of our research include:

MR/AMR Model. We are developing a data model for hierarchical multiresolution (MR) and adaptive multiresolution (AMR) data representations that supports interactive exploration of scientific data. This includes a model of error and error operations that helps keep the experimenter informed of the quality of data at various resolutions. Also, we characterize the kinds of operations that can be performed on MR/AMR data, especially as they relate to data in a distributed computing environment.

Geometry and Topology. Our data model distinguishes between the geometry and topology of a dataset, allowing us to characterize a wide variety of data types. This work should allow us to develop a taxonomy of scientific data that helps exploit regularities in both geometry and topology. We see the topology as a bridge between the scientist's geometric data view and the index-oriented view of the underlying database.

Lattice Model. The lattice model is a single-level model of data that incorporates our ideas about geometry and topology, and is an important component of the formal model especially for representing adaptive resolution data. Geometry and topology forms the basis of a lattice class hierarchy that can efficiently represent a variety of scientific data.

Domain Representation. We are developing an efficient way to represent domains, and especially the extent of subdomains within an enclosing domain. The representation should be space efficient and quick to access. This work is particularly important because we represent certain metadata (e.g., extracted features, classifications) as labeled subdomains.

Data Storage. We must examine how to store the data points in the underlying database. Our approach is spatially coherent, i.e., given a point, we have efficient database access to its geometric neighbors.

Evaluation. The model will be evaluated by implementing a prototype and testing its performance with large datasets. Our goal is a system that is flexible and expressive enough to be of real assistance as the scientist works with the data.

2 Data Model Foundations

Pfaltz et al. [Pfaltz98] identify the major features of scientific data as large size, complex entities and relationships, and volumetric retrieval. However, we need a more rigorous definition if we hope to provide effective database support for scientific data. For our purposes, scientific data is a collection of values that represents some natural phenomenon. This phenomenon is a function over a multidimensional domain. The notion of a dataset *domain* is central to our model of scientific data. The *value space* of the

function defined over the domain usually consists of the cartesian product of the value ranges of several data attributes. This is equivalent to saying that any point in D has a number of attributes — the value of the data function at that point. Each data value says something about a particular point in the domain.

We focus on scientific data that can be represented in a continuous k-dimensional data space [Cigno97]. If a data set consists of *some* attributes that are ordinal, independent, and defined on a continuous value range, the data set contains *dimensional data*, and those attributes are *dimensional.*

2.1 Geometry, Topology, and Neighborhoods

The terminology used in the literature to describe various systems of grids is not standardized. We are developing a more comprehensive and consistent framework for describing and defining grids that encompasses most reported grid structures, including both point and cell data organizations. We separately represent the underlying space in which the grid is defined, which we call the *geometry*, and the relationships implied by the grid, which we call the *topology*. Thus, the *geometry* of a data set refers to the space defined by the dimensions; the *topology* of a data set defines how the points of the grid are connected to each other. A data set's topology is a graph with data points as nodes and arcs between nodes representing a *neighbor* or *adjacency* relationship.

This approach enables database support for application algorithms to process data either geometrically or topologically. In many cases, the topology and/or geometry do not have an explicit representation within the data set because they derive easily from the indexes of an array that stores the data points. The array and its index structure compose the *computational space* of a data set.

2.2 Error

Most scientific data contains some inherent error. This includes measurement error from sampling or computational error from simulation. Furthermore, operations and analyses may introduce additional error. Our model of scientific data includes *localized error* (i.e., it is estimated at every point within the domain [Wong95a]) as well as cumulative error.

2.3 Data Representation

Effective exploration tools for very large data sets are best developed on top of a rigorous conceptual model of the data. Such a model must be accessible to both the programmer and the user and must be able to adapt to the actual data in a natural and efficient way. Our data model that represents a promising foundation for describing scientific data that can be organized into a multiresolution hierarchy.

A rigorous definition of a *data representation* is the formal basis of our model of the scientist's data set. Although the data set represents a phenomenon defined over a continuous domain (a *geometric* space with an infinite number of points), the data set is a finite sampling of this space. Consequently, our *data representation* is defined over a finite set of *sample points*, within the domain. A data representation has several components that define the domain, sample points, and value space of the data. See [Berg00] for more detail.

2.4 The Lattice Representation

Although the data representation definition is comprehensive enough to encompass most kinds of scientific data, it only represents the actual data and does not incorporate any notion of how the different data elements might be related in a grid structure. We incorporate the grid definitions into our data model by adopting and extending the *lattice* which includes a topology, as well as a data representation. Lattices are discussed in more detail in [Berg00].

2.5 Simple Data Model

Our notions of *data representation* and *lattice* are sufficient to represent a gridded scientific data set, but not the phenomenon that the data set is intended to model. Our notion of a *data model* uses the lattice to approximate the phenomenon in the domain, as well as the error. The *approximating function* is normally based on the sample points and returns a value that approximates the phenomenon in the domain.

3 Multiresolution Data Model

Although the basic data model described above represents a very wide range of basic data sets, it is not adequate as a model for multiresolution data. A multiresolution (MR) model allows a researcher to view data using resolutions ranging from very coarse to very fine (the original data). Using a coarse resolution can vastly reduce the size of the data that needs to be stored, manipulated, and displayed. It also serves as an overview of the entire dataset, allowing the researcher to pick out regions of interest without examining the original data directly. Once an interesting region has been identified, the researcher may examine it at finer resolutions, perhaps even accessing the original data. "Drilling down" allows the researcher to examine only data of interest at fine resolution, minimizing processing and display costs.

3.1 Multiresolution Data Representation

The MR representation offers a tradeoff between detail and efficiency. Incorporating multiple resolution capability into the data model allows the database system to provide direct support for managing and using data at the resolution most appropriate to the immediate task.

A *reducing operator* transforms one data model into another data model, where the new representation is smaller than the old [Cigno97]. This reduction introduces additional associated localized error which

must be modeled. An MR hierarchy is formed by repeated applications of reducing operations. The process is repeated a number of times until the size of the data has been reduced sufficiently or until further reductions would introduce too much error. Certain classes of wavelet functions form an ideal basis for reducing functions because of their localized error characteristics [Wong95a], but our model is also appropriate for very different kinds of data reduction techniques such as triangle mesh simplification [Cigno97].

3.2 Adaptive Multiresolution (AMR)

An *adaptive resolution (AR) representation* allows resolution to vary within a single lattice. The resolution near a point may depend on the behavior of the sampling function, on local error, or on the nature of the domain in the neighborhood of the point. A reducing operator that behaves differently over parts of D can define an *adaptive multiresolution representation* (AMR), which is an MR hierarchy in which each layer is an AR representation. For example, it can reduce resolution in areas with lowest error when forming the next level. It might also try to preserve resolution in areas of rapid value change and reduce resolution in less volatile areas. Because an AMR contains multiple resolutions within each level, it has the potential to achieve a representation with the same accuracy as MR using less storage. Alternatively, for a given amount of memory, it can retain increased detail and accuracy in important regions of the domain.

3.3 Support and Influence

Our model for MR is very general. In practice, most MR hierarchies are defined entirely by operations on the sampling set, and they often place further restrictions on a reducing function such as requiring spatial coherence. Typically, any neighboring set of sample points S_j in λ_i should map to a neighboring set of sample points S_k in λ_{i+1}. S_j forms the *support* for S_k as shown in figure 1. For any point p there is a set of points in the next level that claim p as part of their support. We call this set of points the *influence* of p (see figure 1). By building the notions of *influence* and *support* explicitly into the data model (and into the database support system), we can provide a framework for better implicit support for efficient data distribution and distributed computation.

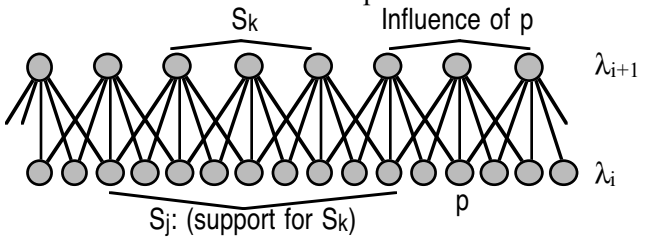


Figure 1. Support and Influence

4 Taxonomy of Geometry and Topology

Since our representation of data must be efficient, it is worthwhile to categorize the way that data points lie in the geometry, and how they are connected in the topology. Our classification is motivated by the desire to exploit patterns within the spacing of the sample points, so the data can be represented efficiently. The taxonomy must be able to represent both cell and point based grids and transformations between them such as Delaunay and Voronoi techniques. Our software design for the lattice representation follows from this taxonomy. We are particularly interested in how much information must be stored in order to describe the geometry and topology of the dataset.

4.1 Periodic Tilings and Data

The study of tilings (tessellations) has some relevance to our research since topologies often define a tiling. A review of this field can be found in [Schatt97].

If a tiling is *periodic*, then it is possible to duplicate the tiling, translate it some distance, and place it down again so that it matches exactly with the original copy. That is, the tiling consists of a number of translated repetitions of some pattern of tiles. An important and related property of periodic tilings is that there exists a subset of the space S that can be repeatedly copied and translated throughout the space to complete the tiling. A minimal subset of this kind is called a *fundamental domain* or *generating region*.

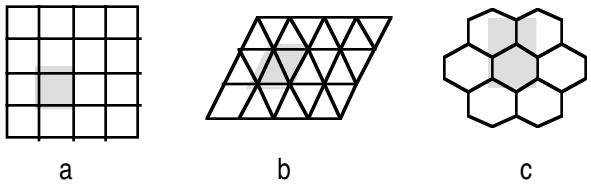


Figure 2. The fundamental domains of the 2D regular tilings.

A *regular tiling* is a periodic tiling made up of identical regular polygons [Schatt97]. The three tilings shown in figure 2 are the only regular tilings for 2D space.

This approach does not explicitly distinguish between the geometry and topology components of a grid. The definition of a tiling includes aspects of topology but most concepts are geometry specific. We are adapting these ideas to our work with geometry and topology.

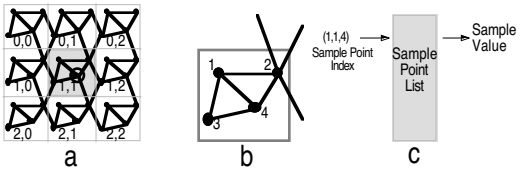


Figure 3. A possible supercell implementation.

We use the notion of the *supercell* to represent periodic sampling topologies. As shown in figure 3, a supercell represents a generating region for the topology, allowing the entire topology to be conceptually represented by a grid of repeated supercells while only storing a single supercell definition. If we can find where in the grid a point lies, we can very easily form a search key from the position in the grid (i.e. supercell identifier), and the position of the point within the supercell (i.e. point identifier). Such a technique promises a quick way to access a point's data given its geometric position.

4.2 Regular Data

Usually, for scientific data to be considered regular, it must lie within a mesh of squares or cubes, perhaps displaced by a shear operation [Cigno97]. By this definition, data points arranged in a hexagonal fashion, as in case *c* of figure 2, would not be considered regular, though the first two would. However, this may only be the case because of the ubiquity of array storage. Researchers tend to think of regular data as any data that can be stored very easily in an array.

It should be possible to develop a rigorous definition of regular data. Besides being regular in the mathematical sense, the patterns in figure 2 have an interesting property: if we store the vertices (sample points) in an array, it is possible to map a point's array indices to its locations in both the geometry and topology without using any other information. Since this property is of immediate interest to designers of scientific databases, it might serve well as a definition of regular scientific data.

4.3 Classifying Irregular Data

With irregular data, it is not possible to map array indices to a location in geometric space without using extra information, if at all. Of course, arrays may still be used to merely store the data points. Figure 4 shows examples of irregular data. In figure 4.a, there is no way to map indices to a geometric location without referring to the spacing between the rows and columns, which varies for each row and column. Therefore, the mapping between indices and geometry must take this spacing as another parameter, i.e. as "extra information". The situation in figure 4.b is even worse. Here, there is no pattern whatsoever to the position of the sample points, so a simple mapping from indices to geometry is out of the question.

We further classify irregular data according to how much information is required to represent the pattern of sample points within the domain. In Figure 4.a, points are lined up in rows and columns, so we only need to store the spacings for each row and column. The space required to store this information is proportional to the number of rows plus the number of columns. In figure 4.b, there is no pattern whatsoever to the sample points, so we must store coordinates for each individual point. Here, the space required is proportional to the number of points. Of course, it is possible to have datasets that combine these attributes, behaving in different ways along different dimensions.

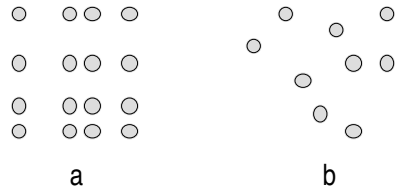


Figure 4. Irregular data

5 Conclusion

We have developed a preliminary data model for describing distributed multiresolution scientific data sets. The model is intended to be the basis for a scientific data management support environment that can provide nearly transparent access to a multiresolution data set regardless of the MR algorithm used.

Bibliography

[Berg00] R. Daniel Bergeron, Philip J. Rhodes, Ted M. Sparr, "A Data Model for Distributed Multiresolution Visualization", *Proceedings of the Dagstuhl Workshop on Scientific Visualization (To Appear), May,* 2000

[Cigno97] Paolo Cignoni, Claudio Montani, Enrico Puppo, Roberto Scopigno, "Multiresolution Representation and Visualization of Volume Data", *IEEE Transactions on Visualization and Computer Graphics, Volume 3, No. 4,* IEEE, Los Alamitos, CA, 1997

[Pfaltz98] John L. Pfaltz, Russell F. Haddleton, James C. French, "Scalable, Parallel, Scientific Databases", *Proceedings 10th International Conference on Scientific and Statistical Database Management,* IEEE, Los Alamitos, CA, 1998

[Schatt97] Doris Schattschneider, Marjorie Senechal, "Tilings", *Handbook of Discrete and Computational Geometry,* CRC Press, Boca Raton, 1997

[Wong95a] Pak Chung Wong, R. Daniel Bergeron, "Authenticity Analysis of Wavelet Approximations in Visualization", *Proceedings of IEEE Visualization '95,* IEEE Computer Society Press, Los Alamitos, CA, 1995

Geometric Fairing of Irregular Meshes using Mesh Hierarchies

Robert Schneider, Leif Kobbelt and H.P. Seidel

Max-Planck Institute for Computer Science, Saarbrücken

1 Introduction

In a recent paper we presented a new fairing algorithm for irregular meshes that form a manifold, based on solving a non-linear fourth order partial differential equation (PDE) that only depends on intrinsic surface properties instead of being derived from a particular surface parameterization. This continuous PDE has a (representation-independent) well-defined solution which we approximate by our triangle mesh. Hence, changing the mesh complexity (refinement) or the mesh connectivity (remeshing) leads to just another discretization of the same smooth surface and doesn't affect the resulting geometric shape beyond this. Our construction algorithm creates a mesh sequence by iteratively updating the vertices until the *outer* and *inner* fairness conditions are sufficiently satisfied, where the *outer* fainess determines the mesh geometry and the *inner* fainess the distribution of the vertices within the surface. To simplify the computation we factorize the fourth order PDE into a set of two nested second order problems thus avoiding the estimation of higher order derivatives.

When applying such a nonlinear fairing algorithm directly to large meshes various problems can occur. Large meshes often contain noise that has to be presmoothed before the Fairing iteration can start since the curvature discretization and the vertex update steps assume an already smooth surface. Moreover, a large mesh has a high vertex density which dramatically decreases the convergence rate of fairing algorithms, especially for nonlinear higher order schemes.

An elegant solution to such problems is to use multigrid techniques during the construction process, based on a fine-to-coarse hierarchical mesh representation. Here the necessary hierarchy levels are constructed using Hoppe's progressive mesh approach with half-edge collapses. However, instead of reducing a mesh while trying to keep the details, we are more interested in creating a mesh whose smallest edge length is maximal while avoiding distorted triangles (long triangles with small inner circle). We start with the construction of a discrete solution on the coarsest level of the progressive mesh representation and then each solution on a coarse level serves as starting point for the iteration algorithm on the next finer hierarchy level. A presmoothing step thus only has to be applied on the coarsest level, later at each hierarchy level the mesh is already presmoothed by the multigrid fairing concept. Between two hierarchy levels we need a prolongation operator that introduces new vertices using the vertex split information of the progressive mesh.

2 Our fairing concept

Following the set-up presented in [8] to define fair surfaces satisfying G^1 boundary constraints, the PDE that determines our geometric fairness concept in this paper is defined as

$$\Delta_B H = 0, \tag{1}$$

which can be interpreted as one possible nonlinear analogon to thin plate splines. Here Δ_B is the Laplace Beltrami operator and H the mean curvature. The PDE only depends on geometric intrinsics and is comparatively simple for a forth order equation. Because of the mean value property of the Laplacian, it is guaranteed that the extremal mean curvature values of a solution of (1) will be reached at the border and that there are no local extrema in the interior. Since constant mean curvature surfaces satisfy this equation, important basic shapes as spheres, cylinders and minimal surfaces satisfying $H = 0$ can be reconstructed.

3 Notation

We partition the vertices of a mesh M into two classes, denoting the set of all border vertices with $V_B(M)$ and the set of all vertices in the interior of M with $V_I(M)$. For each vertex q_i of M let $N(q_i)$ be the set of vertices q_j that are adjacent to q_i and let $D(q_i) = N(q_i) \cup \{q_i\}$ be its 1-disk. Let $H_i = H(q_i)$ denote the discrete mean curvature at the vertex q_i.

4 Construction algorithm

We will now present a short description of the construction algorithm for a mesh M_S that is a discrete solution of equation (1), i. e. it satisfies the outer fairness criterion

$$\Delta_B H(q_i) = 0 \quad \forall\, q_i \in V_I(M_S) \tag{2}$$

plus an additional inner fairness criterion. The input data for our algorithm consists of vertices and unit normals that form the G^1 boundary conditions and an initial mesh M^0 that interpolates the boundary vertices. The idea of the construction algorithm is to create a mesh sequence $M^k, k = 0, 1, 2, \ldots$ by iteratively updating the vertices, until the outer and inner fairness conditions are sufficiently satisfied.

Instead of solving a fourth order problem directly, we factorize it into two second order problems which are solved sequentially. The factorization idea is inspired by the following observation: Given a fixed Laplace-Beltrami operator, fixed mean curvature values at the boundary vertices $V_B(M^k)$ of a mesh M^k and a fixed set of interior vertices $q_i^k \in V_I(M^k)$, (2) can be interpreted as a Dirichlet problem for the H_i, where the unknown scalar mean curvature values at the inner vertices are determined by a nonsingular linear system with a symmetric and positive definite matrix. Solving the resulting nonsingular linear system yields scalar values $\tilde{H}_i$ at all inner vertices $q_i \in V_I(M^k)$, that represent a discrete harmonic function. The idea is now to use this calculated scalar values $\tilde{H}_i$ to update each inner vertex q_i such that $H(q_i^{k+1}) = \tilde{H}_i$, which is again a second order problem. Expressed in two formulas, this factorization of $M^k \to M^{k+1}$ becomes

$$\left.\begin{array}{lrcl} I. & \Delta_B \tilde{H}_i & = & 0 \\ II. & H(q_i^{k+1}) & = & \tilde{H}_i \end{array}\right\} \quad \forall\, q_i^k \in V_I(M^k)$$

We determine the Laplace-Beltrami operator and the boundary mean curvature values by calculating the according values of the current mesh M^k. In practice, it is not necessary to solve the Dirichlet problem exactly. When we have determined the linear system, we apply some iteration steps of an iterative linear solver, using the current mean curvature values as starting values.

5 Multigrid approach

It is well known that the convergence of mesh fairing algorithms can be dramatically accelerated if multigrid techniques are integrated into the construction process [7, 4]. In our implementation we followed this idea. The necessary hierarchy levels are constructed using the progressive mesh approach [5] with half-edge collapses [6]. The number of hierarchy levels can be specified by the user. However, instead of reducing a mesh while trying to keep the details, we are more interested in creating a mesh whose smallest edge length is maximal while avoiding distorted triangles (long triangles with small inner circle).

Our multigrid algorithm exploits the fact that a coarse mesh already approximates the shape of the smooth surface that is implicitly defined by the PDE. Increasing the mesh size mainly improves the smoothness of the approximation (Fig 1). Therefore, we start with the construction of a discrete solution on the coarsest level of the progressive mesh representation

and then each solution on a coarse level serves as starting point for the iteration algorithm on the next finer hierarchy level. Between two hierarchy levels we need a prolongation operator that introduces new vertices using the vertex split information of the progressive mesh. When adding a new vertex q_i, we have to take care that the outer fairness is not destroyed at that position. This is achieved in three steps, where the first two steps are similar to the prolongation operator used by Guskov et al. [4]:

- First we update the mesh topology and introduce q_i at the position given by its inner fairness criterion, which is defined by local discrete Laplacians

$$q_i = \sum_{q_j \in N(q_i)} \lambda_{ij} q_j$$

- In some cases the first step is not enough to avoid triangle distortions, therefore in the second step we further update the complete 1-ring of q_i. This means we solve the local linear problem $\Delta q_l = 0$ for all $q_l \in D(q_i)$.

- Since the second step disturbs the outer fairness, we finally solve $\Delta_B H_l = 0$ for all $q_l \in D(q_i)$ by applying the construction algorithm locally on the 1-disk $D(q_i)$.

Our construction algorithm assumes that the mesh is not a noisy surface. Therefore, before starting the multigrid algorithm we first construct at the coarsest level the solution of the problem based on discretizing the equation $\Delta^2 f = 0$. Later at each hierarchy level our mesh is already presmoothed by the multigrid fairing concept.

References

[1] Bloor, M. I. G., and M. J. Wilson, Using partial differential equations to generate free-form surfaces, Computer Aided Design, 22 (1990), 202–212.

[2] Burchard, H. G., J. A. Ayers, W. H. Frey, and N. S. Sapidis, Approximation with Aesthetic Constraints, in *Designing Fair Curves and Surfaces*, ed. N. S. Sapidis, SIAM, Philadelphia, 1994.

[3] Chopp, D. L., and J. A. Sethian, Motion by Intrinsic Laplacian of Curvature, Interfaces and Free Boundaries 1, 1–18, 1999.

[4] Guskov, I., W. Sweldens, and P. Schröder, Multiresolution Signal Processing for Meshes, SIGGRAPH 99 Conference Proceedings, 325–334.

[5] Hoppe, H., Progressive meshes, SIGGRAPH 96 Conference Proceedings, 99–108.

[6] Kobbelt, L., S. Campagna, and H-P. Seidel, A General Framework for Mesh Decimation, Proceedings Graphics Interface '98, Morgan Kaufmann Publ., 43–50.

[7] Kobbelt, L., S. Campagna, J. Vorsatz, and H-P. Seidel, Interactive Multi-Resolution Modeling on Arbitrary Meshes, Proceedings of SIGGRAPH '98, 105–114.

[8] Schneider, R., and L. Kobbelt, Generating Fair Meshes with G^1 Boundary Conditions, Proceedings of GMP 2000, 251–261.

[9] Welch, W., and A. Witkin, Free-Form shape design using triangulated surfaces, SIGGRAPH 94 Conference Proceedings, 247–256.

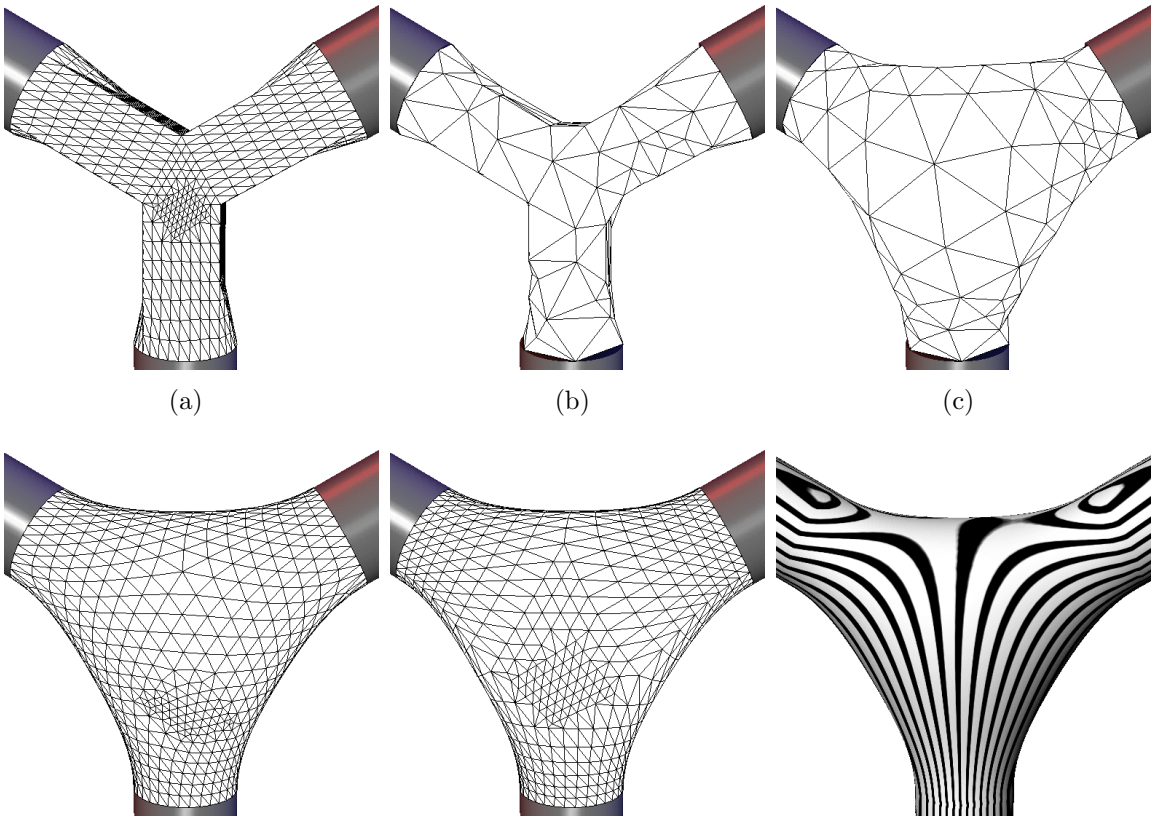


Figure 1: Mesh fairing based on discretizing $\Delta_B H = 0$. The boundary condition is determined by 3 cylinders that are arranged symmetrically. a) shows the original mesh (920 vertices) and b) a reduced version with 118 vertices. In c) and d) we optimized the inner fairness with respection to a local uniform parameterization. In e) we chose a inner fairness condition that produces a mesh that is discrete conformal to the original mesh a). The influence of the mesh size and the vertex distribution is small, a fact that is exploited during the multigrid construction process.

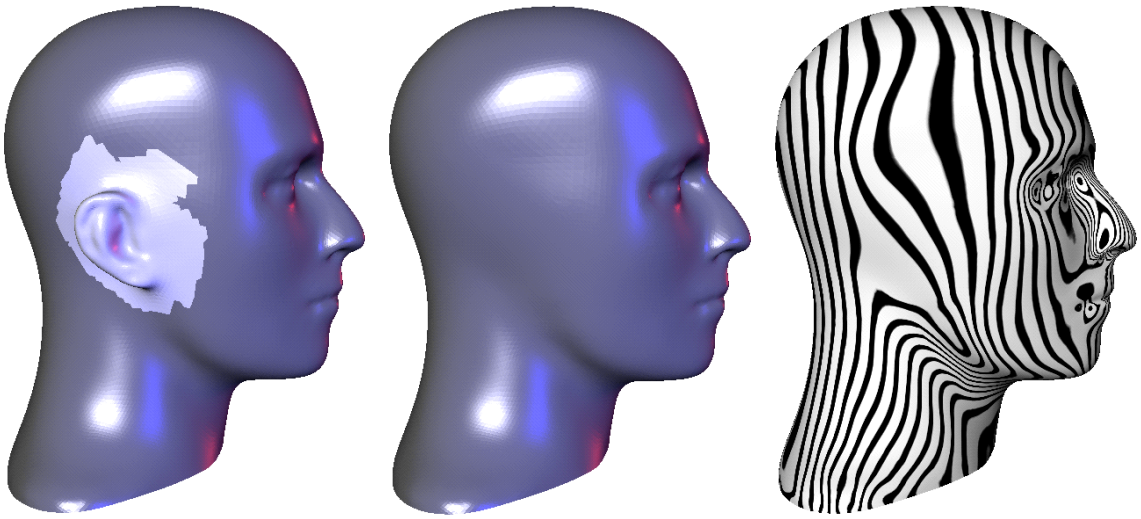


Figure 2: Feature removal of a mesh with user defined boundary curve.

Hierarchical LIC for Vector Field Visualization

Udeepta Bordoloi and Han-Wei Shen
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
E-mail: bordoloi@cis.ohio-state.edu and hwshen@cis.ohio-state.edu

Abstract

This paper presents a hierarchical algorithm to accelerate 2D LIC computation. A quadtree data structure, combined with vector field simplification metrics, are employed to provide the capability of selective LIC approximation. In the algorithm, each node of the quadtree is associated with a measure of "complexity" corresponding to the local flow field. At run time, a threshold is provided by the user to determine the degree of approximation. We report work in progress aiming to solve two fundamental problems: (1) Find an appropriate metric as a measure of the degree of vector field complexity. (2) Develop an approximate LIC algorithm to produce an image that gives a faithful representation of the vector field, i.e., it should have as much information as a normal LIC image.

1 Introduction

One of the most popular methods for vector field visualization is Line Integral Convolution, or LIC [1]. Since this method was first introduced in 1993, researchers have proposed many extensions to improve the computation speed [2, 3], to produce better LIC images [4, 5, 6], and to apply LIC to both steady and unsteady flow fields [7, 8, 9]. The popularity of LIC mainly comes from its effectiveness in depicting the flow directions everywhere in a dense vector field. The disadvantage of LIC, however, is that it is computationally expensive. Even though computing power has increased significantly in the last decade, the amount of data generated from numerical simulations has also increased by many orders of magnitude. In this paper, we present an algorithm that utilizes a hierarchical scheme to accelerate LIC computation, and to make interesting features of a vector field stand out against the relatively uninteresting background.

The main cost of computing LIC images is streamline advection. To reduce the computation time, previously Stalling and Hege have proposed a fast-LIC method [2] which performs convolution incrementally for pixels along a streamline to reduce the overall number of streamlines being computed. In addition, adaptive step size control for a Runge-Kutta numerical integration method is also employed by the fast-LIC method to further speed up the computation. In contrast with the fast-LIC method, the technique presented in this paper adopts a different approach. Instead of computing accurate streamlines everywhere to cover the entire field, streamlines are computed approximately and selectively based on the underlying vector field features. In parts of the vector field where the flow directions are fairly straight or similar across a local region, only one streamline is computed and the same streamline is used by the points in the entire neighborhood to perform convolution. Our goal is to reduce the number of streamlines computed and simplify the LIC computation, and thus reduce the total computation cost.

To make the algorithm suitable for applications that have different visual quality and interactivity requirements, it is very important that different levels of approximations to be provided and controlled. To achieve this goal, we use a *quadtree* hierarchical data structure to provide the capability of selective approximation when computing 2D LIC images. In our algorithm, each node of the quadtree is associated with a measure of "complexity" corresponding to the local flow field. At run time, the user provides a threshold to specify the minimum complexity level to displayed. If the measure associated with a given node is lower than the user supplied value, the corresponding flow field region is simplified and approximate LIC is performed in the region. Otherwise, the algorithm subdivides the region into smaller parts and performs the same test recursively. To make possible such a hierarchical LIC computation, two key issues need to be addressed:

(1) Find an appropriate metric as a measure of the degree of complexity for the local areas of the underlying vector field.

(2) Develop an approximate LIC algorithm to produce an image that gives a faithful representation of the vector field, i.e., it should have as much information as a normal LIC image.

In the following, we present our hierarchical LIC algorithm addressing the above two problems. We first briefly overview the LIC algorithms. We then describe the error metrics that are used in the quadtree hierarchical data structure. Two approximate methods for LIC computation are discussed, and experimental results demonstrating both the image quality and computational speed of the algorithm are presented.

1.1 Related Work

The Line Integral Convolution method is a texture synthesis technique that can be used to visualize two-dimensional vector field data. Taking as the input a vector field and a white noise image with the same resolution as the vector field, LIC computes convolution of the input noise image using the following algorithm: For each pixel, streamlines in both the positive and negative directions are first calculated. The pixel's convolution result is computed by weighted-averaging the image values of the pixels along the streamline paths. As a result, the intensity values of the pixels along each streamline are strongly correlated so the directions of the flow field can be clearly visualized.

While LIC is effective in visualizing 2D vector fields, it is quite computationally expensive. Stalling and Hege proposed an extension to speed up the process [2]. Their work is based on two key observations. First, a streamline starting from any point in the domain actually passes through many pixels. For those pixels, only one rasterized streamline is sufficient to produce the convolution and thus redundant numerical integrations can be avoided. The second observation is that adjacent pixels along the same streamline use very similar sets of pixel values for the convolution. Therefore, the LIC value computed for one pixel can be reused by its neighbors with small modifications to accelerate the convolutions. By reducing the number of streamlines computed and speeding up the LIC convolution, Stalling and Hege's new method can gain a great saving in computing the LIC.

The main idea of this paper is to further speed up the LIC computation by adopting vector field simplification methods and hierarchical data structures [10, 11]. While the remaining of the paper discusses our algorithm for accelerating the standard LIC method, we believe that combining our algorithm with the fast-LIC algorithm can be very effective.

1.2 Measure for simplification

To allow different levels of approximations, we need to provide error measures to characterize the degree of *complexity* in the regions of the vector field. We use the term "complexity" to measure the parallelism between the streamlines, i.e., how the flow directions in a local region are similar to each other. We have implemented two measures to represent the complexity of the vector field. One is the magnitude of *curl*, and the other is the metric proposed by Heckel *et al.*[10]. Our goal is to make regions with features like vortices and saddle points have a high degree of complexity. Uninteresting parts such as straight flows, on the other hand, are considered to have a low degree of complexity.

The measures are calculated for each point in the field at a preprocessing stage, and a quadtree is built out of the information. During the actual LIC image computation stage, we traverse the quadtree in a depth first manner. We adopt the convention that a low value of the measure implies that the region can be simplified, and a high value suggests that there might be interesting features in the region, and we should subdivide the region further, i.e., go down the current quadtree level. Thus the measure we are using denotes 'complexity'. We do not simplify a region even if only one point within it has a 'complexity' value higher than the user defined threshold. This is achieved by keeping track of the maximum value of 'complexity' for each region, and it is this value that is kept at each node of the quadtree.

1.2.1 Curl

The curl at a point in a vector field is defined as the cross product between the divergence of the vector field at that point and the vector given by the field at that point. Mathematically, it has the form:

$$Curl(x, y) = (\frac{dF_y}{dx} - \frac{dF_x}{dy})\mathbf{z}$$

where F_x and F_y are the x and y components of the vector. A intuitive geometrical interpretation of curl at a point would be that it gives a measure of how much the vector field curls around that point. Thus the magnitude of curl at the center of a whirlpool would be very high, and that of a point in the middle of a straight flow would be zero. The quadtree preprocessing for curl is straightforward. We first calculate the magnitude of curl for each point in the vector field, and then build the quadtree over the vector field in a bottom up fashion. Each node of the tree stores the maximum value of curl in the region the node represents.

The magnitude of curl represents a local measure of complexity for the flow, as only the vectors from a point's adjacent points are used in the calculation. A very low curl at a point means that the flow is almost parallel at that point. But it would not necessarily mean that the flow is parallel some distance from the point. Potentially, this nature of locality can become problematic when simplifying the LIC computation as the streamline convolution path originating from each point has a much wider span. Some experimental results are presented and discussed in the result section.

1.2.2 Streamline Distance Error

Another flow complexity measure that we chose to use was proposed by Heckle *et al.*[10]. For each point, the metric represents the difference between the accurate and the approximate streamlines originating form the point in question. More specifically, the error is measured as the sum of distances of the corresponding points on the two streamlines, as shown in Figure 1.

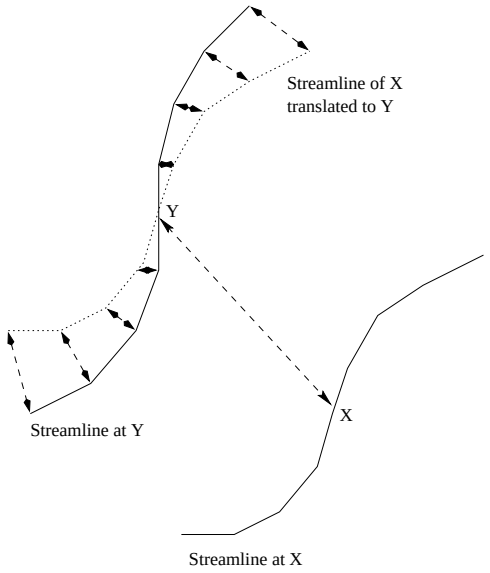


Figure 1: Streamline distance error metric

Generating a quadtree out of this measure is slightly different from the maximum quadtree method that we use for the curl metric. At the lowest level of the quadtree, say a 2×2 block, we calculate the error at each point as the distance between the actual streamline originating from that point and the approximate streamline, which is a translated streamline computed form the center of the 2×2 block. The leaves store the maximum of the error distances of the four points in the 2x2 block. For the next level, we follow the same steps for the 4x4 region a node represents. This time, the value at the node is the maximum of the error distances of the sixteen points in the region, and the error distances are computed with respect to the streamline of the center point of this region. Note that this center point is different from any of the center points of the 2x2 sub-regions of this region.

To approximate a LIC image, this metric is believed to be more intuitive if we are to translate an approximate streamline across a region to perform the convolution. If the translated streamline does not model the actual streamline well, the distance would be high and thus we need to use a finer level of quadtree approximation. Before we present the approximation result using this metric, we first we discuss our LIC approximation methods in the next section.

1.3 Approximating LIC

For a region that has a low complexity measure, we intend to limit the calculation of streamlines as much as possible, i.e., use only one (or possibly a few) streamline to approximate the flow directions for the whole region. In the following we describe two methods for approximating LIC using the quadtree-based hierarchical vector field simplification measure that we have discussed.

1.3.1 Streamline Translation

Our first method is to translate an approximate streamline to each point of a local region and use the approximate streamline trajectory as the LIC convolution path. For a region that is determined to

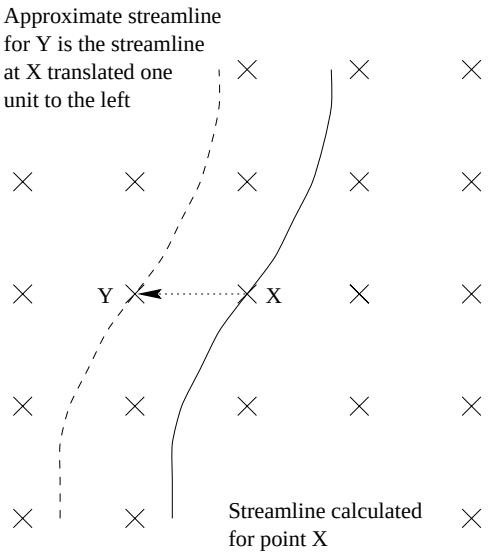


Figure 2: LIC approximation using streamline translation

be simplified, we select one point, near the center of the region, and calculate the actual streamline originating from this point and perform rasterization of the streamline. To calculate LIC for any other point in the region, we translate the rasterized streamline from the center point by a displacement vector (the vector from the central point to the current point that we are approximating the streamline for), and use it to perform the same convolution as the standard LIC.

Figure 2 shows an example of our algorithm. Consider the grid point that is to the left of the current point. Since we assume that the shape of the streamlines is similar, we just translate the current streamline one unit to the left to get the approximate streamline for the new point.

1.3.2 Pseudo LIC

We have also experimented with Pseudo LIC (PLIC) proposed by Verma *et al.*[12], which uses texture mapping to generate LIC-like textures. PLIC places seed points for streamlines on a uniform grid, which might be sparser than the grid of the output image. (Compare this to the traditional method of LIC, in which seed points for streamlines are pixels of the output image). To generate flow textures, a rectangular LIC texture of a straight field is warped and texture mapped onto the streamline. The quality of PLIC output depends on how sparse the seed points are, and how wide and how long the rectangular patch used for texture map is. More details can be found in [12].

In our algorithm, the streamline is calculated at the central point of the region to be simplified. Instead of performing regular LIC convolution for each point in the region, we texture map a rectangular patch from a previously calculated LIC image of a straight vector field onto the streamline. The catch is that the seed points are no longer uniformly spaced, and we need to calculate a proper width and length for the rectangular patch depending on the size of the region we are simplifying.

2 Results and Discussion

In the following, we first give qualitative assessments on the merits and demerits of the metrics we use, and then discuss the speedups and the result images.

We have applied our algorithm to a 400×400 two dimensional vector field with three vortices and two saddle points. Figures 3-6 show results using the streamline translation method for LIC approximation with the curl and the streamline distance as the error metrics. Figures 7-10 were generated using PLIC approximation with the same set of error metrics. Figure 11 is the traditional LIC image for comparison. The streamline distance error metric is calculated using a streamline advected to a length of 40 units. The images generated using the streamline translation method use a convolution length of 40 units, while the Pseudo LIC images are convolved to 20 units.

From the figures we note that when the curl is used as a metric, the image begins to show noticeable artifacts around the vortices. The reason for this can be better understood from the image of the magnitude of the curl of the vector field in Figure 12. The curl is very high at the saddle points and also at the centers of the vortices. But as we move away from the vortex centers, the curl falls to levels that are comparable to areas with a straight flow. This happens because curl is calculated locally, while the streamline calculated for generating the LIC images spans a much greater region. Thus one of the drawbacks of curl is that it fails to capture the global features to the field.

The streamline distance error metric is very well suited for the translation method of approximating streamlines. Since the error is calculated using the same translated streamline that is later used for approximation, it tells us exactly how good or bad the approximation is. The effects of simplification when using the streamline distance error metric, however, start to appear near the saddle points. From the images of the various levels of the distance error in Fig 13 and 14, we notice that the error is high for the vortices but really low around the saddle points.

Based on the above observations, neither curl nor the streamline distance error appear to be the perfect metric for vector field simplification on their own right. It is likely that a combination of both the curl and the streamline distance would prove to be much more faithful in representing the 'complexity' of regions of vector fields.

Using either of the LIC approximation methods, appreciable speedup can be achieved (see Figure captions) with different degrees of convolution artifacts. The tradeoff between image quality and computation speed is controlled by the user. The streamline translation method produces artifacts as the coherence becomes weaker between the pixels that are adjacent to each other but in different regions of the quadtree simplification. In addition, the correlation between the pixels in the region that is simplified is also disturbed as the convolution paths for the pixels that are along the same streamline in the original field now become slightly different as a result of the streamline approximation. However, for the flow regions that have low complexity, the differences are small in general. Pseudo LIC seems to be a much better candidate for approximation. We do not see discontinuity along the boundaries of the regions that are simplified. In the original Pseudo LIC algorithm, streamlines are uniformly spaced. For our purposes, we have to place them according to the quadtree traversal, which results in non-uniform placement. This causes different parts of the final image to have different intensities because they have different amounts of texture mapped values deposited on them. We solve this problem by starting new streamlines from points which do not have a minimum number of texture mapped deposits on them.

3 Future Work

There is scope for more work on both the error metrics and the LIC approximation methods. We believe we can produce a better metric by combining more than one property of the vector field, in this case the curl and the streamline distance. As far as approximation is concerned, we can use concepts of fast-LIC in our algorithm

to achieve higher speedups. Finally, the use of a hierarchical algorithm should prove very useful for producing three dimensional LIC images, where the challenge is both computational speed and selective display of information.

Acknowledgments

This work was supported by The Ohio State University Research Foundation Seed Grant. We would like to thank Ravi Samtaney for providing the test data sets and Dr. Roger Crawfis for useful comments.

References

[1] B. Cabral and C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 93*, pages 263–270. ACM SIGGRAPH, 1993.

[2] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 95*, pages 249–256. ACM SIGGRAPH, 1995.

[3] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel line integral convolution. In *Proceedings of First Eurographics Workshop on Parallel Graphics and Visualisation*, pages 111–128, September 1996.

[4] M.-H. Kiu and D. Banks. Multi-frequency noise for LIC. In *Proceedings of Visualization '96*, pages 121–126. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[5] H.-W. Shen, C.R. Johnson, and K.-L. Ma. Visualizing vector fields using line integral convolution and dye advection. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 63–70. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[6] V. Interrante and C. Grosch. Strategies for effectively visualizing 3d flow with volume lic. In *Proceedings of Visualization '97*, pages 421–424. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[7] L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, 1995.

[8] A. Okada and D. L. Kao. Enhanced line integral convolution with flow feature detection. In *Proceedings of IS&T/SPIE Electronic Imaging '97*, pages 206–217, 1997.

[9] H.-W. Shen and D.L Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2), 1998.

[10] B. Heckel, G. Weber, B Hamann, and K. Joy. Construction of vector field hierarchies. In *Proceedings of Visualization '99*, pages 19–25. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[11] A. Telea and J. van Wijk. Simplified representation of vector fields. In *Proceedings of Visualization '99*, pages 35–42. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[12] V.. Verma, D. Kao, and A. Pang. Plic: Bridging the gap between streamlines and lic. In *Proceedings of Visualization '99*, pages 341–348. IEEE Computer Society Press, Los Alamitos, CA, 1999.

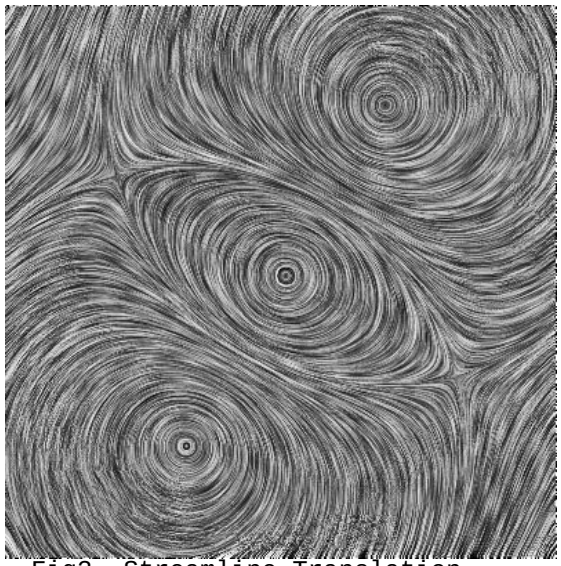


Fig3. Streamline Translation, metric:curl, speedup:21.2%

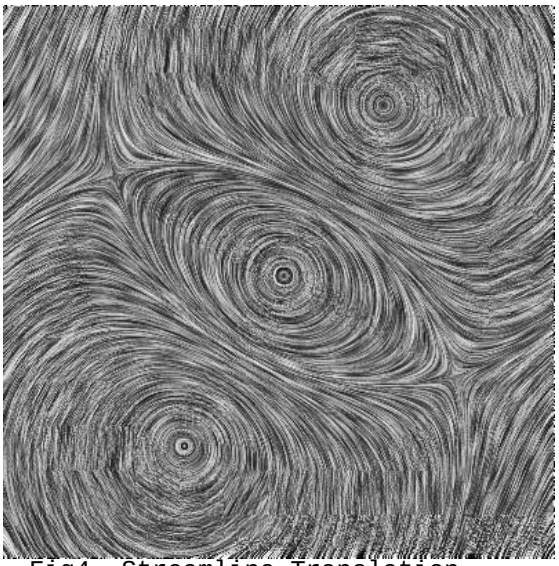


Fig4. Streamline Translation, metric:curl, speedup:50.3%

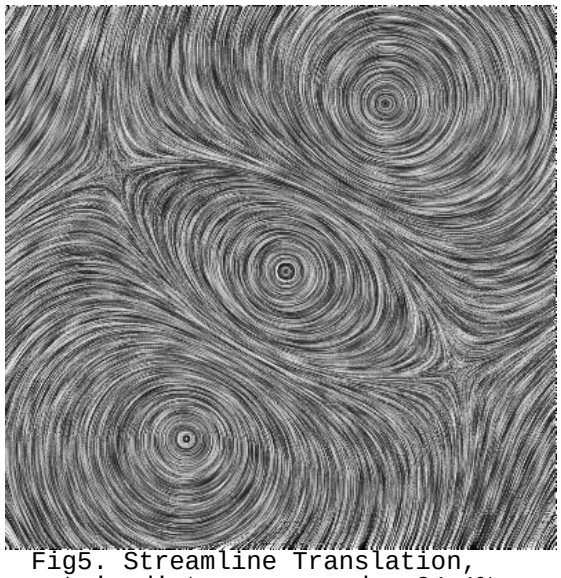


Fig5. Streamline Translation, metric:distance, speedup:34.1%

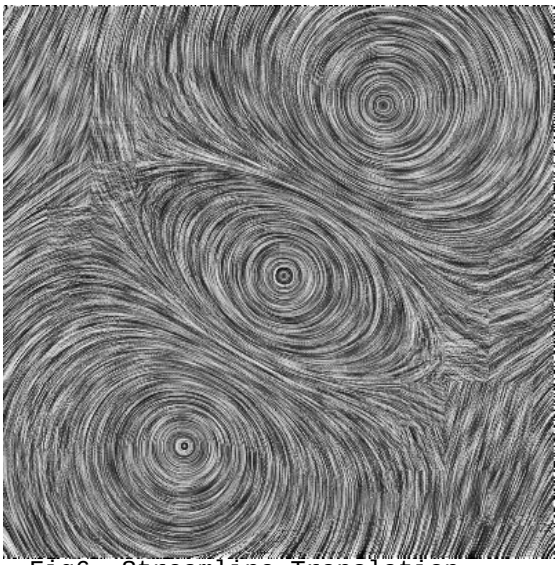


Fig6. Streamline Translation, metric:distance, speedup:52.6%

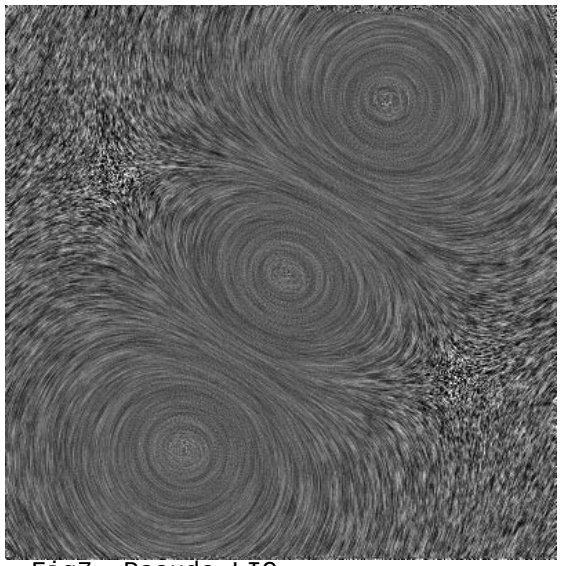


Fig7. Pseudo LIC, metric:curl, speedup:23.6%

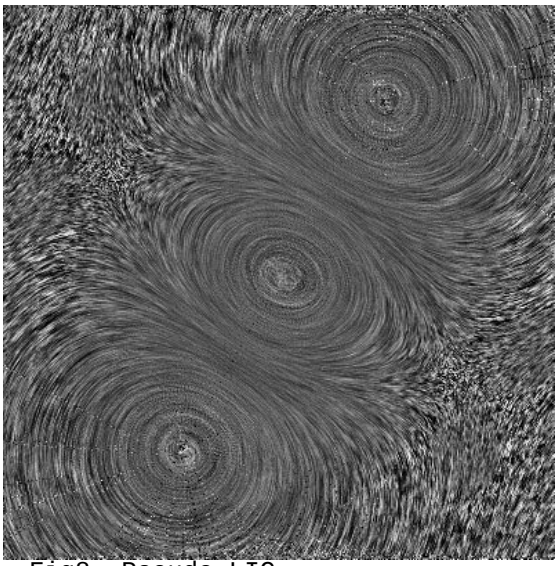


Fig8. Pseudo LIC, metric:curl, speedup:43.3%

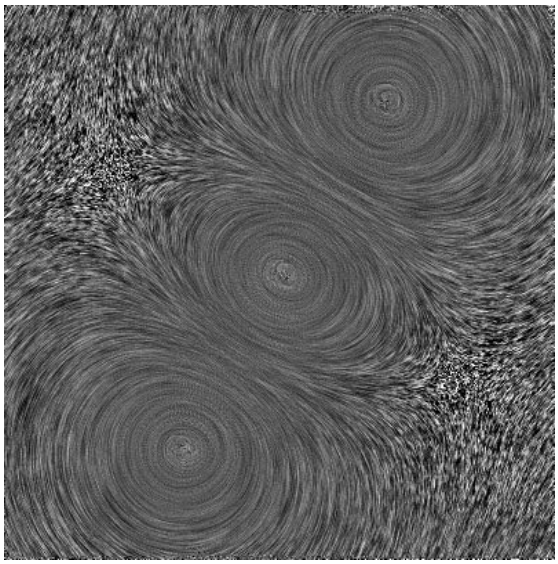


Fig9. Pseudo LIC,
metric:distance, speedup:25.3%

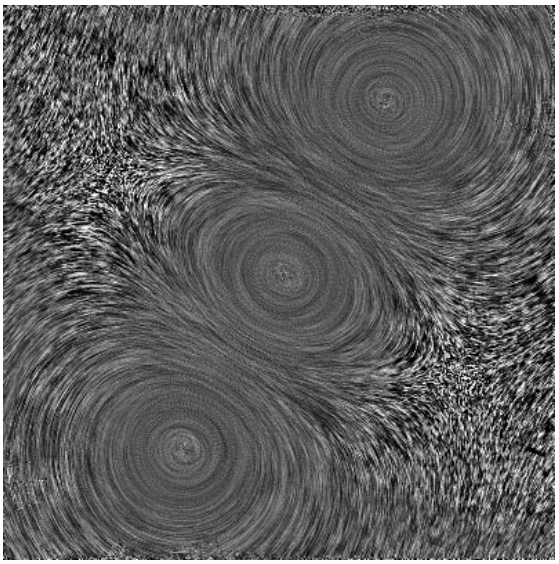


Fig10. Pseudo LIC,
metric:distance, speedup:47.5%

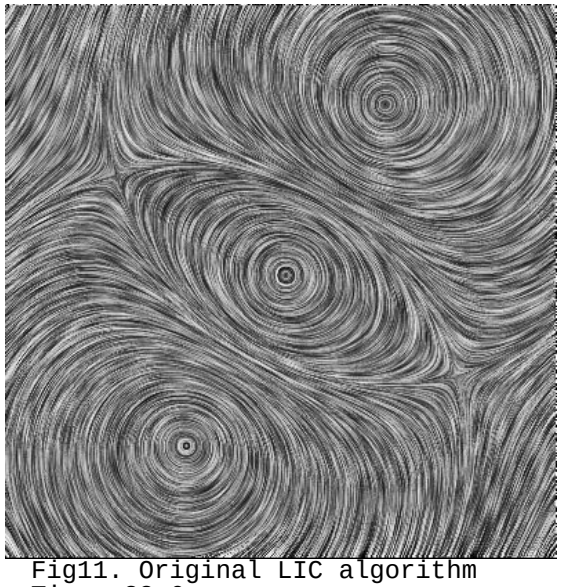


Fig11. Original LIC algorithm
Time: 33.6s

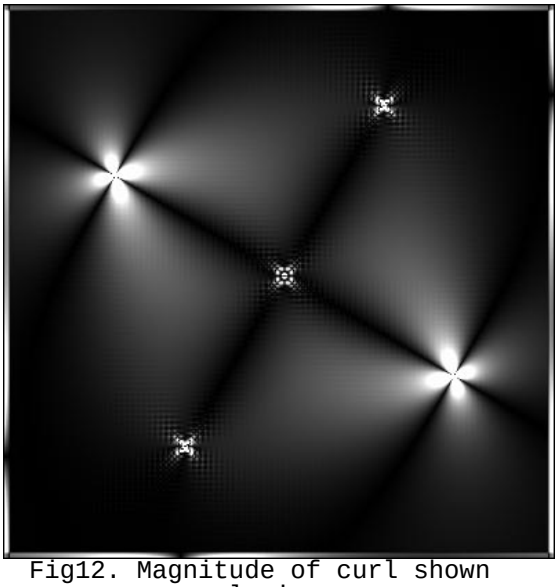


Fig12. Magnitude of curl shown
as a gray scale image

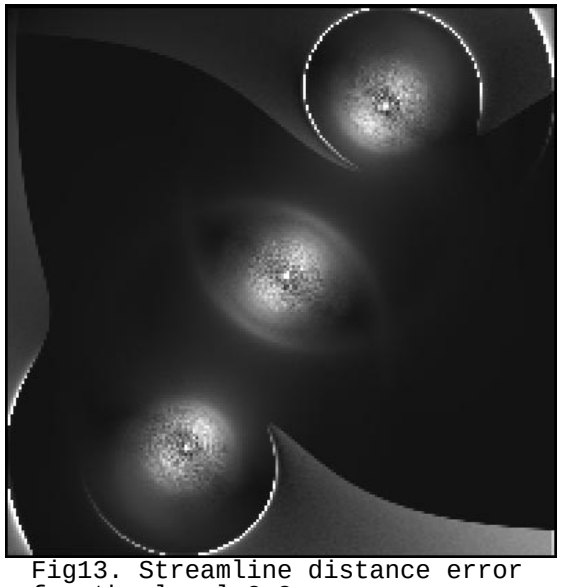


Fig13. Streamline distance error
for the level 2x2

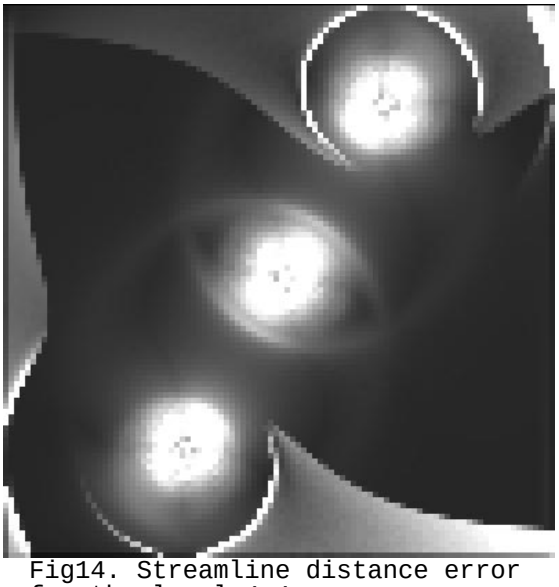


Fig14. Streamline distance error
for the level 4x4

Approximating Material Interfaces during Data Simplification

David E. Sigeti* Benjamin F. Gregorski†
John Ambrosiano * Gerald Graham *
Mark A. Duchaineau‡ Bernd Hamann † Kenneth I. Joy †

* Los Alamos National Laboratory
† University of California, Davis
‡ Lawrence Livermore National Laboratory

1 Introduction

We present a new method for simplifying large data sets that contain material interfaces. Material interfaces embedded in the meshes of computational data sets are often a source of error for simplification algorithms because they represent discontinuities in the scalar or vector fields over a cell. By representing material interfaces explicitly in a data simplification process, we are able to provide separate field representations for each material over a single cell and, thus, to represent the fields much more accurately. Our algorithm uses a multiresolution tetrahedral mesh supporting fast coarsening and refinement capabilities and error bounds for feature preservation. We represent a material interface or other surface of discontinuity as the zero set of a signed distance function. This representation makes it possible to maintain continuity of the surface across cell boundaries. It also makes it possible to represent more complex interface structures within a cell, such as T-intersections. Within a cell, a field is represented on either side of the surface of discontinuity by separate linear functions. These functions are determined by true and "ghost" values of the field at the vertices of the cell. By requiring that each vertex have only one ghost value for a given field and material, we are able to avoid introducing spurious discontinuities in the fields at cell boundaries. The use of linear functions determined by ghost values makes it unnecessary to divide the original cells in the mesh along the surface of discontinuity, avoiding the resultant introduction of complex cell types and field representations. It also decouples the field representation from the representation of the surface of discontinuity, making it easier to represent fields when the material interfaces are more complex. Both the signed distance function that defines the surface of discontinuity and the ghost values that determine the field representations are handled very simply during refinement and coarsening of the mesh ensuring that all spurious discontinuities can be avoided with a minimum of computation and programming effort. We have applied our algorithm to simplification of a test problem from a well known fluid dynamics code with excellent results. Graphical and numerical results are presented. Furthermore, our multiresolution representation can be applied to other kinds of surfaces, e.g. isosurfaces.

2 Multiresolution Tetrahedral Mesh

As the geometric basis for our simplification algorithm we use the subdivision of a tetrahedral mesh presented by Zhou et al [1]. This framework has an important advantage over other multiresolution spatial data structures such as an octree—it makes it easy to avoid introducing spurious discontinuities into our representations of fields. The way we perform the binary subdivision ensures that the tetrahedral mesh will always be a conformant, i.e, all edges in the mesh end at the endpoints of other edges and not in the interior of edges. The simplest representation for a field within a tetrahedral cell is just the unique linear function that interpolates field values specified at the cell's vertices. In the case of a conformant mesh, this natural field representation will be continuous across cell boundaries, resulting in a globally C^0 representation.

We have generalized the implementation presented by Zhou et al by removing the restriction that the input data needs to be given on a regular rectilinear mesh consisting of $(2^N + 1) \times (2^N + 1) \times (2^N + 1)$ points. A variety of input meshes can be supported by interpolating field values to the vertices of the multiresolution tetrahedral mesh. In general, any reasonable interpolation procedure may be used. In some cases, the procedure may be deduced from the physics models underlying the simulation that produced the data set. In other cases, a general-purpose interpolation algorithm will be appropriate.

We construct our data structure as a binary tree in a top-down fashion. Data from the input data set, including grid points and interface polygons, are assigned to child cells at the time that their parent is split.

The other basis for our algorithms is the ROAM system, described in [2]. ROAM uses priority queue-driven split and merge operations to provide optimal real-time display of triangle meshes for terrain rendering applications. The tetrahedral mesh structure used in our framework can be regarded as an extension to tetrahedral meshes of the original ROAM data structure for triangle meshes.

Since our data structure is defined recursively as a binary tree, a representation of the original data can be computed in a preprocessing step, and we can utilize the methods developed in ROAM to efficiently select a representation that satisfies an error bound or a desired cell count. This makes the framework ideal for interactive display.

Strict L^∞ error bounds are incorporated into the subdivision process, see Section 5 below.

3 Representing Material Interfaces

In the class of input datasets with which we are working, material interfaces are represented as triangle meshes. In the case that these triangle meshes are not known, they are extracted from volume fraction data by a material interface reconstruction technique described in [3] and [4] (The

*{ambro,ggraham,sigeti}@lanl.gov
†{gregorsk,hamann,joy}@cs.ucdavis.edu
‡{duchaineau1}@llnl.gov

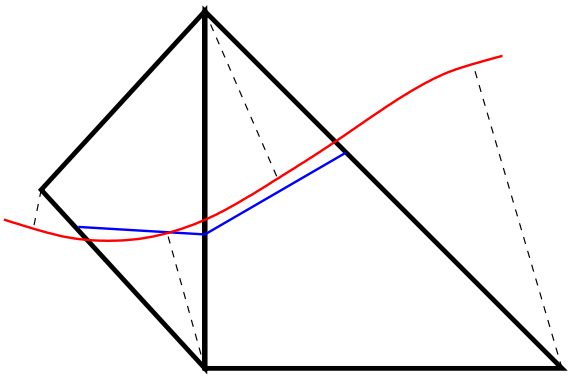


Figure 1: True and approximated interfaces.

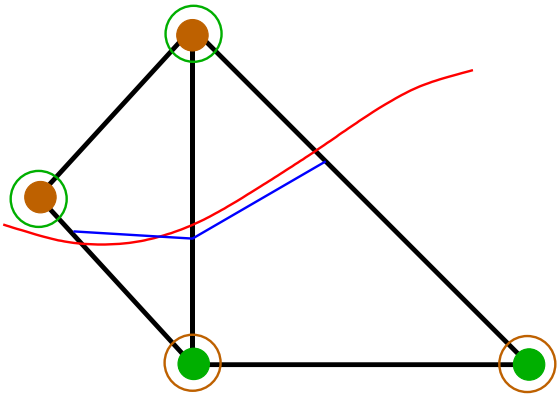


Figure 2: Ghost values.

volume fractions resulting from numerical simulations indicate what percentages of which materials are present in each cell.). Such an interface reconstruction technique produces a set of crack-free triangle meshes and normal vector information that can be used to determine on which side and in which material a point in space lies.

Within one of our tetrahedra, an approximate material interface is represented as the zero set of a signed distance function. Each vertex of a tetrahedron is assigned a signed distance value for each of the material interfaces in the tetrahedron. This value is simply the minimum distance from the vertex to the interface. The sign of the distance is given by the side of the interface on which the vertex lies.

Figure 1 shows a two-dimensional example of two triangles forming a conformant mesh, crossed by an interface (shown in red). The minimum distances from the vertices of the triangles to the interface are shown as dotted lines. The distances for vertices on one side of the interface (say, above the interface) are assigned positive values and those on the other side are assigned negative values. These signed distance values at the vertices will then determine linear functions in each of the triangles and the approximated interface (shown in blue) will be the zero set of these linear functions. Because the mesh is conformant, the linear functions in the two triangles will agree on their common side, and the zero set will be continuous across the boundary. The situation in three dimensions is analogous, with the word "triangle" replaced by "tetrahedron".

We note that, in order for the interface representation to be continuous across cell boundaries, it is necessary both that the mesh be conformant and that each vertex have at most one signed distance value for each interface.

The signed distance values for a vertex are computed when the vertex is created in a split operation. When searching for the point on the true interface that is closest to the vertex, it is possible to restrict attention those cells that share the edge being split. This makes the computation very efficient for the great majority of vertices.

4 Representing Discontinuous Fields

Once we have approximated the interfaces within a cell, we must decide how to represent fields on either side of the interface. Our algorithm represents the discontinuity by constructing a linear field representation for each material in the cell. In order to specify these representations, each of the vertices in a cell must have a distinct field value for each material in the cell. When a vertex does not lie in a given material, the field value associated with that material is called a *ghost value*.

The use of ghost values is illustrated in Figure 2. The material on the upper side of the interface is represented by brown and the material on the lower side is represented by green. The two upper vertices lie in the brown material and, thus, have regular values for the field in the brown material. These values are indicated by the the solid brown circles. The empty green circles indicate that these vertices require ghost values for the green material. Similarly, the lower circles lie in the green material and, thus, have regular values for the field in the green material (solid green circles) and require ghost values for the field in the brown material (empty brown circles). Once we have such ghost values, we can define linear representations for the field in the two regions by the usual interpolation. If we maintain a conformant mesh and assign only a single ghost value for a given material to a vertex, then our field representation will be discontinuous where it should be (across the interface) but not across cell boundaries (which would be a spurious discontinuity). Once again, the situation in three dimensions is analogous, with the word "triangle" replaced by "tetrahedron".

In our current implementation, we choose as the ghost value for a given vertex, field, and material the value of the field at the point in the material that is closest to the cell. These points are, of course, exactly the points that were used to determine the distance map that defines the approximation to the interface.

The ghost values for a vertex are computed when the vertex is created during the tetrahedral refinement process.

5 Error Bounds and Refinement Strategy

The error bounds employed in our framework are similar to the nested error bounds used in the ROAM system. Each cell has two associated kinds of error values, field errors and material interface errors.

Field errors are first calculated for leaf cells and are then propagated up the hierarchy. So far, we have only worked with input data sets that may be considered to consist of discrete grid points. In this case, the computation of error bounds for leaf cells is straightforward—the error for a leaf cell is simply the maximum of the errors associated with all the grid points from the input data set that it contains. When fields in the input data set are considered to have

values over finite volumes, the computation of leaf cell errors will be more complex.

The field error e_T for a non-leaf cell is computed from the errors associated with its two children according to:

$$e_T = \max\{e_{T_0}, e_{T_1}\} + |z(v_c) - z_T(v_c)| \quad (1)$$

where e_{T_0} and e_{T_1} are the errors of the children; v_c is the vertex that splits the parent into its children; $z(v_c)$ is the field value assigned to v_c; and $z_T(v_c)$ is the field value that the parent assigns to the spatial location of v_c, equivalently, $z_T(v_c) = \frac{1}{2}(z(v_0) + z(v_1))$, where v_0 and v_1 are the vertices of the parent's split edge. This error bound is *nested* in the sense that the error of a child is guaranteed not to be greater than the error of the parent.

The material interface error associated with a leaf node is the maximum of the errors associated with each of the interfaces in the node. For each interface, the error is the maximum distance between the approximate representation of the interface in the cell and those polygons that define the true interface and which are contained in the cell.

We initially refine our mesh to meet a user-determined error bound on the location of interfaces. The mesh is then further refined, using the ROAM algorithms, to minimize the error in a given field consistent with a given tetrahedron count.

6 Results

We have tested our algorithm on a data set resulting from a simulation of a hypersonic impact between a dense projectile and a less dense metal block. The simulation uses a logically rectilinear mesh of dimensions 32x32x52. For each cell, the density and pressure values are available, as well as the per-material densities and volume fractions. The physical dimensions in x, y, and z directions are [0,12] [0,12] and [-16,4.8].

There are three materials in the simulation: the projectile, the block, and empty space. The interface between the projectile and the block consists of 38 polygons, the interface between the projectile and empty space consists of 118 polygons and the interface between empty space and the block consists of 17574 polygons.

Figures 3 shows a cross section view of the mesh created by a cutting plane. The black lines are the original interface polygons intersected by the plane, and the magenta lines are our approximation to the interface. The interface approximation error is 0.15. An error of 0.15 means that all of the vertices in the original material interface meshes are no more that a physical distance of 0.15 from their associated interface approximation. This is equivalent to an error of (0.5 - 1.5)% when considered against the physical dimensions. A total of 3174 tetrahedra were required to approximate the interface to an error of 0.15. The overall mesh contained a total of 5390 tetrahedra. A total of 11990 tetrahedra were required to approximate the interface to an error of 0.15 and the density field to an error of 3. The maximum field approximation error in the cells containing material interfaces was 2.84 and the average field error for these cells was 0.007. These error measurements indicate that separate field representations for the materials on either side of a discontinuity can accurately reconstruct the field.

Figures 4 and 5 compare density fields generated using linear interpolation of the density values to fields generated using separate field representations on either side of the discontinuity. Figure 5 shows that using explicit field representations in the presence of discontinuities can improve the quality of the field approximation. This can be seen in the flat horizontal and vertical sections of the block where the cells approximate a region that contains the block and empty space. In these cells, the use of explicit representations of the discontinuities leads to a very accurate representation of the density field. The corresponding field representations using linear interpolation, shown in Figure 4, do a very poor job of capturing the discontinuities. Furthermore, Figure 5 captures more of the dynamics in the area where the projectile is entering the block (upper left corner). The linear interpolation of the density values in the region where the projectile is impacting the block smoothes out the density field, and does not capture the distinct interface between the block and the projectile. Figure 6 shows the density field from Figure 5 with our approximation to the interface and without the cell outlines.

7 Conclusions and Future Work

We have presented a simplification method for scientific data sets that explicitly represents material interfaces in mesh cells. Our algorithm constructs an approximation that can be used in place of the original data set for visualization purposes. Explicitly representing the material and implicit field discontinuities allows us to use multiple field representations to better approximate the field within each cell. The use of the tetrahedral subdivision allows us to generalize our algorithm to a wide variety of data sets and to support interactive level-of-detail exploration and view-dependent simplification. Future work will extend our error calculations to support complex input cell types such as tetrahedra and curvilinear hexahedra. Our current ghost value computation assumes that the field is constant on the other side of the interface. Higher-order extrapolation methods should be investigated for ghost value computation to determine if a superior field approximation can be obtained. Similarly, material interfaces are defined by approximations based on linear functions. The tradeoff between cell count and higher-order approximation methods should be investigated to determine if a better approximation can be obtained without a great increase in computational complexity. Finally, we plan to apply our algorithm to more complex unstructured data sets.

References

[1] Y. Zhou, B. Chen, and A. Kaufman, "Multiresolution tetrahedral framework for visualizing regular volume data," in *IEEE Visualization '97* (R. Yagel and H. Hagen, eds.), pp. 135–142, IEEE Computer Society Press, 1997.

[2] 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 *IEEE Visualization '97* (R. Yagel and H. Hagen, eds.), pp. 81–88, IEEE Computer Society Press, 1997.

[3] K. S. Bonnell, "On material boundary surfaces," Master's thesis, University of California at Davis, June 2000.

[4] K. S. Bonnell, M. A. Duchaineau, D. R. Schikore, B. Hamann, and K. I. Joy, "Constructing material interfaces from data sets with volume-fraction information," in *IEEE Visualization 2000*, 2000. to appear.

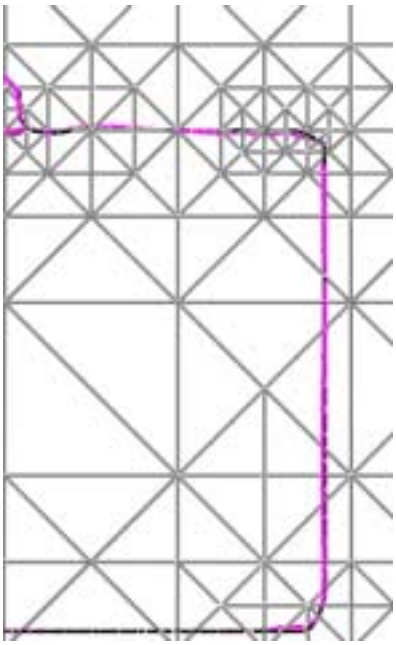


Figure 3: Cross section of the tetrahedral mesh showing the original interfaces and interface approximations.

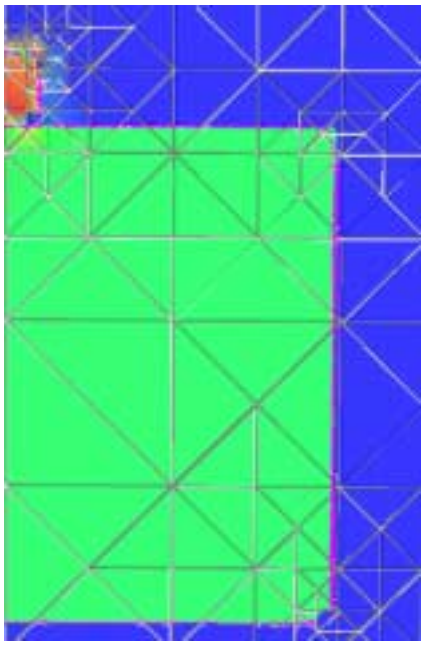


Figure 5: Density field using explicit interface representations and separate field representations (interface error = 0.15).

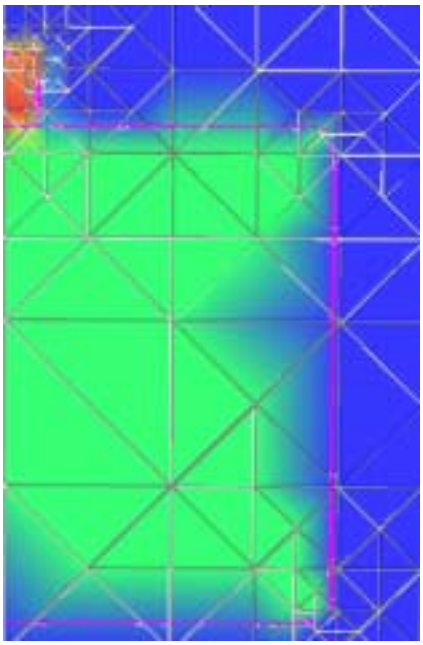


Figure 4: Density field using linearly interpolated density values (interface error = 0.15).

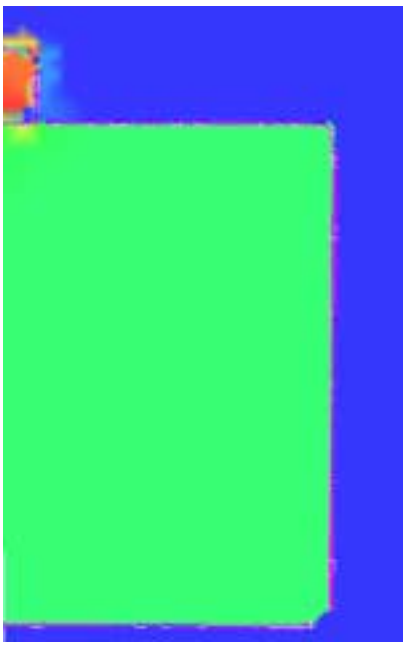


Figure 6: Figure 5 without the cell outlines.

Crest lines extraction from 3D triangulated meshes

Georgios Stylianou and Gerald Farin
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5407

1 Abstract

We present an automatic crest lines extraction method from 3D triangulated meshes. We use a bivariate polynomial to approximate the surface locally and calculate the principal curvatures and directions on every vertex and a tracing algorithm to extract the crest lines. We show crest lines on various triangulated meshes and evaluate their invariance under rotation and stability under scaling.

Keywords:3D triangulation, crest lines, principal curvatures.

2 Introduction

Crest lines are 3D lines on a surface that provide a satisfactory geometrical representation of important physical properties such as anatomical features in the case of medical images [2].

Lengagne *et al* [1], extract crest lines from 3D triangulations. This method has problems regarding the derivative calculation of the largest curvature because it makes a very rough approximation of the derivative that fails in many cases.

Guéziec [3] extracts crest lines using a B-spline parametrization of the skull. Although, such a parametrization is the most accurate, is very expensive and most importantly it can not be used to parametrize every surface.

Khaneja *et al* [2] implemented a dynamic programming algorithm to extract crest lines from triangulations that is stated to have noise immunization. This method is semi-automatic and was applied to medical data only. The problem is it requires the presense of an expert to define start-end points for every crest line; thus this method fails when there is no expert or the expert fails to identify topologically correct points.

We propose a method that solves the problems appearing in [1], is automatic in addition to [2] and can be applied in any surface as long as it is represented by a triangulated mesh in contrast to [3].

This article is organized as follows. In Section 3 we briefly describe the underlying theory of our method. Sections 4, 5 present the method and the experimental results.

3 Theory

In this section, we briefly give the theoretical aspects of this work.

3.1 Crest lines

Crest lines are local shape features of a surface. They are as the set of of all points satisfying

$$D_{\mathbf{t}_1} k_1(u,v) = 0 \qquad (1)$$

where k_1 is the largest principal curvature and $\mathbf{t}_1$ is its direction in the (u,v)-domain.

They are shape features with the main charasteristic of using local information to yield a global description of the surface.

3.2 Parametrization

We use a quadratic bivariate polynomial to parametrize locally the surface.

The patch we use has the type

$$\mathbf{x}(\mathbf{u}) = \mathbf{a}u^2 + \mathbf{b}uv + \mathbf{c}v^2 + \mathbf{d}u + \mathbf{e}v + \mathbf{f} \qquad (2)$$

where $\mathbf{u} = (u,v)$ is a point on the domain, $\mathbf{x}(\mathbf{u})$ is a point on the surface and $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}$ are the coefficients of the patch.

3.3 Least Squares fitting

We use the least squares [4] method to fit (2) on a point set. Here we define the problem and describe the domain calculation.

Problem: Given a point set $D = \{\mathbf{p}_i | i = 0, ..., L\}$ we want to find an approximating patch $\mathbf{b}(u, v)$, such that

$$\sum_{i=0}^{L} \|\mathbf{p}_i - \mathbf{b}(u_i, v_i)\|^2 \quad (3)$$

is minimized.

The first step is the domain calculation i.e. the calculation of the parameters $\mathbf{u}_i = (u_i, v_i)$ that correspond to $\mathbf{p}_i$. In other words, we have to calculate the domain parameters $\mathbf{u}_i$ of the prospective range values $\mathbf{p}_i$. The procedure we use is due to Hamann [6]. Here we rapidly give the steps.
In order to approximate the principal curvatures at a point $\mathbf{x}_i$ do

1. Get the neighboring points of $\mathbf{x}_i$.
2. Compute the best plane P, passing through these points.
3. Define an orthonormal coordinate system in P with a point in D as the origin.
4. Project all the points of D onto the plane P and represent their projections with respect to the local coordinate system in P.

The final points are the wanted domain points. In our case, the plane P is the plane with normal vector the normal vector on a vertex of the triangulated mesh.

The second step is the patch fitting. We solve the linear system

$$D = FX$$

which is equivalent to minimizing 3. D is the given point set, F is the domain values of this point set D and X are the unknowns, the coefficients of the polynomial 2.

4 Method

Here, we describe the method we propose for automatic crest line extraction. The method is twofold. The first step approximates the principal curvatures and directions on every vertex of the triangulation and the second step traces the crest lines.

The first step is:
For every vertex $\mathbf{v}_i$ of the mesh,

1. Get the star of $\mathbf{v}_i$, denoted by $star(\mathbf{v}_i)$.
2. Fit the quadratic patch (2) on $\mathbf{v}_i$ and $star(\mathbf{v}_i)$.
3. Compute the principal curvatures and their corresponding directions on $\mathbf{v}_i$.

Then we evaluate whether a vertex is a crest point. The local neighborhood of a vertex $\mathbf{v}_i$ of a triangulated mesh is its $star(\mathbf{v}_i)$. We can utilize the definition (1) and directly calculate the directional derivative of the largest curvature of every vertex like Lengagne *et al* [1] or using numerical differences but the derivative is very noise sensitive and it fails to provide consistent values many times especially on irregular triangulated meshes. In fact, we have experimented using the numerical differences method and it, often, does not give acceptable results. Consequently, we take a different approach.

We use the interpetation of definition (1) where *a point is a crest point if its largest curvature is locally maximal in its corresponding direction.* Therefore, after calculating the principal curvature k_1^i and its corresponding domain direction $\mathbf{t}_1^i$ of a vertex $\mathbf{v}_i$, we use the domain values of the $star(\mathbf{v}_i)$ as follows:

- The domain values of the $star(\mathbf{v}_i)$ are $(\mathbf{u}_1, ..., \mathbf{u}_L)$, L is the number of vertices in the $star(\mathbf{v}_i)$ and $\mathbf{u}_0$ is the domain value of $\mathbf{v}_i$.
- Find the edge $(\mathbf{u}_0, \mathbf{u}_m)$, $m \in [1, L]$ closest to $\mathbf{t}_1^i$.
- Find the edge $(\mathbf{u}_0, \mathbf{u}_j)$, $j \in [1, L]$ and $j \neq m$ closest to $-\mathbf{t}_1^i$.
- Get the largest curvature of the corresponding vertices k_1^m, k_1^j.
- If $(k_1^i)^2 - (k_1^m)^2 > e$ and $(k_1^i)^2 - (k_1^j)^2 > e$ then $\mathbf{v}_i$ is a crest point.

where $e >= 0$ controls the level of maximality.

The second step of the method is the tracing of crest lines over the mesh. Letthe mesh consist of N vertices. The algorithm is:

1. Initialize linelist L and set the number of lines $j = 0$.

2. Set the id of the current vertex $i = 0$.

3. if $\mathbf{v}_i$ is not visited and is a crest point
   - call *traceCrestLine*($\mathbf{v}_i, l_j$,"first").
   - call *traceCrestLine*($\mathbf{v}_i, l_j$,"last").
   - increase j.

4. If $i < N$ increase i and goto step 3.

The procedure *traceCrestLine*($\mathbf{v}_i, l_j$) is:

1. Mark $\mathbf{v}_i$ as visited and add it to line l_j.

2. Get all the vertices of $star(\mathbf{v}_i)$ that are crest points and are not visited. Let their number be n.

3. If $n = 1$ then goto step 1 for the new vertex, else exit.

When we call *traceCrestLine* with the argument "first" it adds every point on the beginning of the line, otherwise with the argument "last" adds every point on the end of the line. Therefore, we can trace the maximum line segment possible and not just parts of a line.

5 Results

We have tested our algorithm on various meshes. Firstly, we used regular triangulated meshes representing different kinds of vases. Figures 1 and 2 show the crest lines superimposed on vase1 and vase2, respectively. Figure 3 shows the crest lines

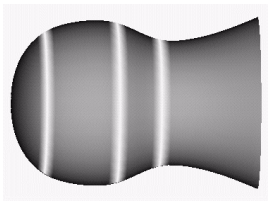


Figure 1: Crest lines of vase1.

for vase2 after scaling two times in x and y coordinates, before extracting crest lines. They are very stable, even though, we streched the object. The only problem on the crest lines is that they are slightly translated from their actual position, as it is visually observed. This problem occurs because we trace crest lines over the vertices of the mesh.

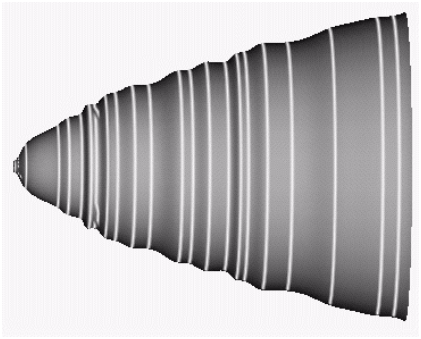


Figure 2: Crest lines of vase2.

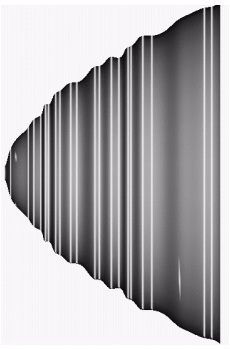


Figure 3: Crest lines of vase2 after streching.

Figures 4 and 5 show the crest lines extracted from the triangulated mesh of the Stanford bunny in two different mesh resolutions. Bunny2 consists of 8171 vertices and 16301 facets. Bunny3, which is a decimated version of bunny2, consists of 1889 vertices and 3751 facets. To obtain these results we set the threshold $e = 3.0$ and $e = 0.5$ for bunny2 and bunny3, respectively. Even though the level of resolution is quite different, we get very similar results. The substantial difference is that the crest points of bunny2 are much more stable than bunny3's in terms of their level of maximality.

Moreover, even with these nice results we have some artifacts. Lines that should not exist or very small lines. These can be removed by deleting the lines that have less than a given number of points. This keeps the most significant lines.

Figure 6 shows the crest lines extracted as in [1]. Clearly, they are very different and too many lines appear that should not exist. The problem that causes all these artifacts is the derivative calculation of the largest curvature.

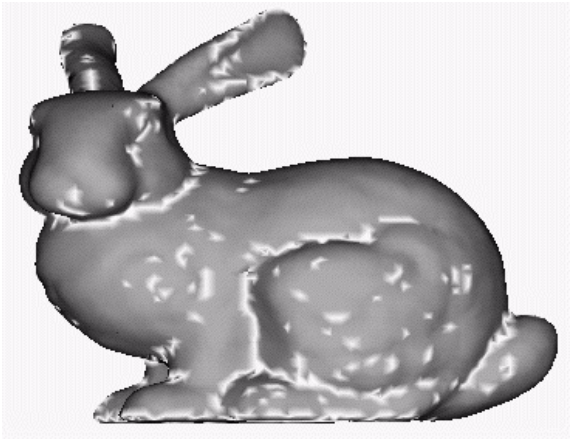


Figure 4: Crest lines for Stanford bunny2.

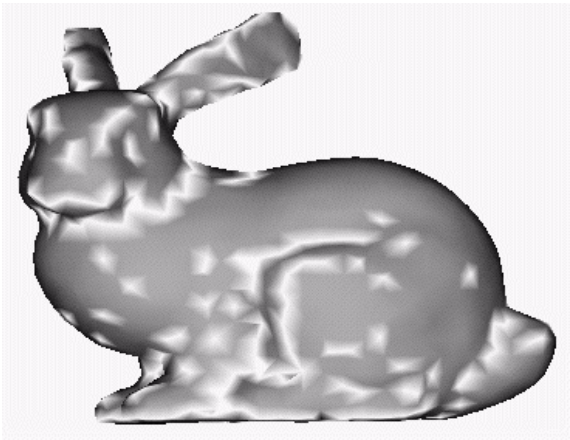


Figure 5: Crest lines for Stanford bunny3.

6 Conclusions

In this work, we have presented a method for crest lines extraction. We used a bivariate polynomial to calculate principal curvatures and directions on every vertex of the mesh and used definition (1) to decide when a vertex is a crest point. Later, we gave an algorithm to trace the crest lines over the mesh and stored them in a useful data structure for future use. Finally, we shown its efficiency on various triangulated meshes, and evaluated the invariance of crest lines under rigid transformations and stability under streching.

In future work, we intend to implement a quantitative comparison method using crest lines, improve noise control and make a parallel implementation of this method for extracting crest lines efficiently on triangulated meshes of order of hundreds of thousands of points.

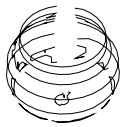


Figure 6: The crest lines of vase1 as in [1]

7 Acknowledgments

This work was supported in part by the Arizona Alzheimer's Disease Research Center.

References

[1] R. Lengagne, F. Pascal, O. Monga. Using Crest Lines to Guide Surface Reconstruction from Stereo. *ICPR '96*,1996.

[2] N. Khaneja, M.I. Miller, U. Grenander. Dynamic Programming Generation of Curves on Brain Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

[3] A. Guéziec. Large Deformable Splines, Crest lines and Matching. *IEEE 4th International Conference on Computer Vision*,650-657,1993.

[4] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design, Fourth Edition.* Academic Press, Boston, 1997.

[5] A. Gray. *Modern Differential Geometry of Curves and Surfaces with Mathematica, Second Edition.* CRC Press,1998.

[6] B. Hamann. Curvature Approximation for Triangulated Surfaces. In *Computing*, 139-153, Springer-Verlag, 1993.

Scaling the Topology of Symmetric, Second-Order Tensor Fields

Xavier Tricoche, Gerik Scheuermann, Hans Hagen, Stefan Clauss
University of Kaiserslautern, Department of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern
Germany
E-mail: {tricoche|scheuer|hagen|clauss}@informatik.uni-kl.de

1 Introduction

Tensors are the language of mechanics. Therefore, tensor field visualization is a challenging issue for scientific visualization. Scientists and engineers need techniques that enable both qualitative and quantitative analysis of tensor data sets resulting from experiments or numerical simulations. A topology-based visualization method of symmetric, second-order tensor fields in two dimensions has been designed for that purpose [1]. It focuses on one of the two eigenvector fields corresponding to the minor or major eigenvalues. Like the vector case, the displayed graph consists of so-called degenerate points (where both eigenvalues are equal) connected by particular integral curves, the separatrices. This technique proved to be suitable for tensor fields with simple structure because the extracted topology contains few degenerate points and separatrices, leading to a comprehensible structure description. However, for tensor fields with complicated structure (like those encountered in turbulent flows, for instance), topology-based methods result in cluttered pictures that confuse the interpretation.

Our hierarchical method merges close degenerate points into one, which simplifies the topology and clarifies its depiction, though globally maintaining the qualitative properties of the original data. It is the extension to the tensor case of a paper we presented at the IEEE Visualization 2000 Conference [2]. We assume a planar, piecewise bilinear interpolated, symmetric, second-order tensor field over a structured grid. The degenerate points are determined first. A clustering strategy is then applied on the resulting set of degenerate points. This leads to a grid partition into cell clusters such that the distance between degenerate points in each cluster is below a user-prescribed threshold. After this, we merge the degenerate points lying in each cluster to get the desired scaling effect. Finally, we determine and integrate the resulting separatrices.

The clustering strategy has been already presented earlier for vector fields [2]: Starting with the whole domain as initial cluster, one computes the distance of the degenerate points to their midpoint. If this distance exceeds a prescribed threshold, the cluster is split into two subclusters along an edge polyline chosen to optimize the distance criterion in the resulting subclusters. The merging process and the analysis of the resulting degenerate points present new aspects and are detailed in the following.

We have applied our method to a swirling jet simulation with evolving turbulence. In this case, the topology is clarified, the separatrices easier to track while the significant structural features have been preserved.

2 Eigenvectors and Eigenvalues

First, we give here the analytic expressions of the eigenvalues and eigenvectors of a symmetric, second-order tensor. We denote the considered tensor field as follows:

$$T := \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}$$

where α, β and γ are bilinear scalar functions in x and y.
Except in the special case where the matrix is diagonal, the system to solve leads to the following expressions. The eigenvectors are $\vec{v}^{\pm} := (v_1, v_2^{\pm})^T$ with

$$v_1 = 2\beta \quad \text{and} \quad v_2^{\pm} = (\gamma - \alpha) \pm \sqrt{(\alpha - \gamma)^2 + 4\beta^2}$$

The associated eigenvalues are (with the same notations)

$$\lambda^{\pm} = \frac{(\gamma+\alpha) \pm \sqrt{(\alpha-\gamma)^2+4\beta^2}}{2}$$

That is, λ^+ is the major eigenvalue associated with eigenvector $(v_1, v_2^+)^T$ and λ^- is the minor eigenvalue associated with eigenvector $(v_1, v_2^-)^T$.

3 Local Topology Deformation

In each cluster, we aim at locally simulating the fusion of all preexisting singularities while preserving consistency with the original global topology. Practically, we must remove the degenerate points, replace them by a single one and let the cluster boundary unchanged. Like in the vector case, we achieve it as follows. We first remove all quadrilateral cells in the cluster. Then we add a new vertex at the cluster center, associated with a degenerate tensor value. At last, we cover the resulting empty domain by linear interpolated triangles joining the new vertex to those on the cluster boundary. Yet, contrary to the vector case, a degenerate tensor value is not unique and may be any *isotropic* tensor (i.e. of the form λI_2). Fortunately, one can suppress the isotropic component of the tensor field (one gets then the so-called *deviator* component) for it does not modify the topology of its eigenvectors (see [3], Theorem 1). That is, one associates a zero matrix value with the new vertex.

4 "Parallel" Positions Localization

Once such an artificial degenerate point has been created, we need to determine its local structure to find the positions of its separatrices. As a matter of fact, separatrices are integral curves that bound the so-called hyperbolic sectors of a singularity (see [2], section 4.2.1). For that reason, boundary curves and type of the sectors must be found. As the singularity lies inside a piecewise linear domain, one can show that its structure may be fully identified on the piecewise linear edges on the boundary of its containing cluster. For each such edge, one has the following configuration: One is given a segment $\overline{[M_1 M_2]}$ with linear parameterization $t \in [0,1]$ on which a linear varying, symmetric, second-order tensor field is defined. Furthermore, one is given the cluster center position Ω that does not lie on $\overline{[M_1 M_2]}$. Now, we seek on $\overline{[M_1 M_2]}$ the positions $M(t)$ where the vector $\Omega\vec{M}$ is parallel to the eigenvectors, that is

$$\Omega\vec{M} \wedge \vec{v} = \vec{0}.$$

The linear parameterization can be written

$$M(t) \;=\; (1-t)\; M_1 \;+\; t\; M_2, \quad t \in [0,1].$$

Thus

$$\Omega\vec{M} \wedge \vec{v} = \Omega\vec{M}_1 \wedge \vec{v} + (1-t)M_1\vec{M}_2 \wedge \vec{v}.$$

With the notations

$$\left\{ \begin{array}{rcl} \Omega\vec{M}_1 & =: & (x_1, y_1)^T \\ M_1\vec{M}_2 & =: & (\delta_x, \delta_y)^T \end{array} \right.$$

and supposing $\beta(t) \neq 0$, the "parallel" condition becomes

$$\left| \begin{array}{cc} x_1 \;+\; (1-t)\;\delta_x & 2\;\beta \\ y_1 \;+\; (1-t)\;\delta_y & (\gamma-\alpha) \;\pm\; \sqrt{(\alpha-\gamma)^2+4\;\beta^2} \end{array} \right| \;=\; 0$$

(α, β and γ are here linear functions in t).
Straightforward calculus leads finally to a cubic polynomial equation in t that we solve with an analytic method. The positions found are the possible locations of separatrices.

5 Sector Type Identification

Because of the sign indeterminacy, the sector type identification cannot be obtained by the method described in the vector case. As a matter of fact, the distinction between hyperbolic and elliptic type is impossible without additional information. For that reason, we use the tensor index defined by Delmarcelle [4]. Between two consecutive "parallel" positions, we compute the angle variation of the considered eigenvector field (see Fig. 1). Depending on the sector type, one gets

- $\Delta\alpha = \theta$ in the *parabolic* case
- $\Delta\alpha = \theta - \pi$ in the *hyperbolic* case
- $\Delta\alpha = \theta + \pi$ in the *elliptic* case

which enables a sector type identification. When a hyperbolic sector is found, its boundary curves are then integrated to form the topological graph.

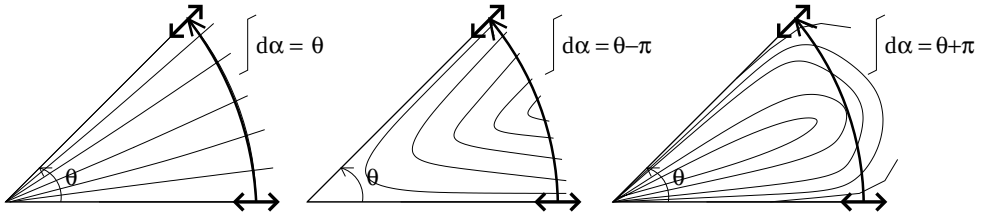


Figure 1: Angle variation in the parabolic, hyperbolic and elliptic case

6 Results

We show next the results of our method applied on the rate of strain tensor field of a swirling jet simulation. The dataset is almost axisymmetric, a property that is quite well preserved by our algorithm. The grid is structured and has 124 x 101 cells ranging from 0 to 9.87 in x, resp. -3.85 to 3.85 in y. The original topology presents 61 degenerate points and 131 separatrices while the topology scaled with threshold 0.4 has 31 singularities and 76 separatrices and the topology scaled with 0.8 has 15 degenerate points and only 42 separatrices. (The degenerate points created artificially are shown in yellow).

Acknowledgment

The authors wish to thank Wolfgang Kollmann, MAE Department of the University of California at Davis, for providing the swirling jet dataset. Furthermore, we would like to thank Tom Bobach, Holger Burbach, Jan Frey, Aragorn Rockstroh, René Schätzl and Thomas Wischgoll for their programming efforts.

References

[1] Delmarcelle T., Hesselink L., *The Topology of Symmetric, Second-Order Tensor Fields.* Proceedings IEEE Visualization'94, 1994.

[2] X. Tricoche, G. Scheuermann, H. Hagen, *A Topology Simplification Method for 2D Vector Fields.* Proceedings IEEE Visualization'00, 2000.

[3] Lavin Y., Levy Y., Hesselink L., *Singularities in Nonuniform Tensor Fields.* Proceedings IEEE Visualization'97, 1997.

[4] Delmarcelle T., *The Visualization of Second-Order Tensor Fields.* PhD Thesis, Stanford University, 1994.

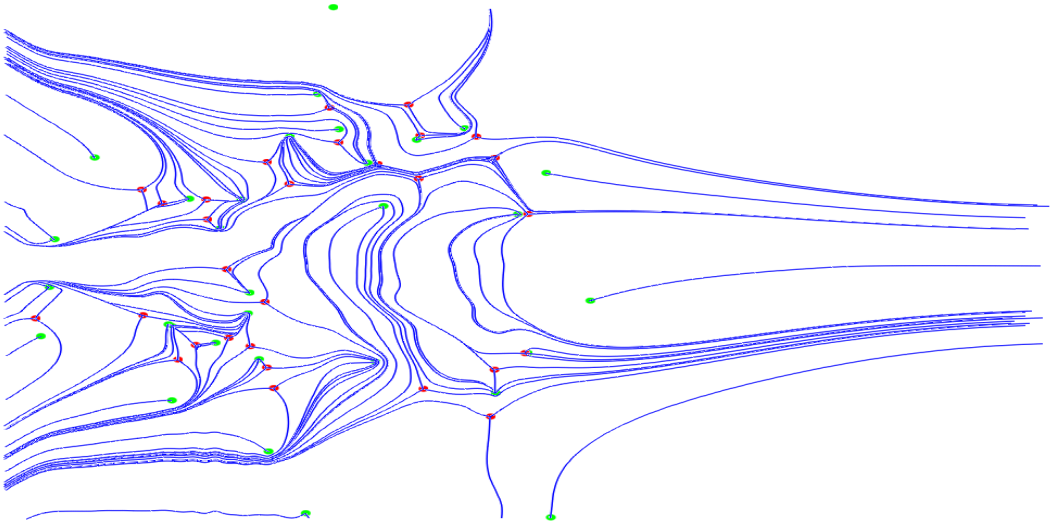


Figure 2: Original topology: 61 degenerate points

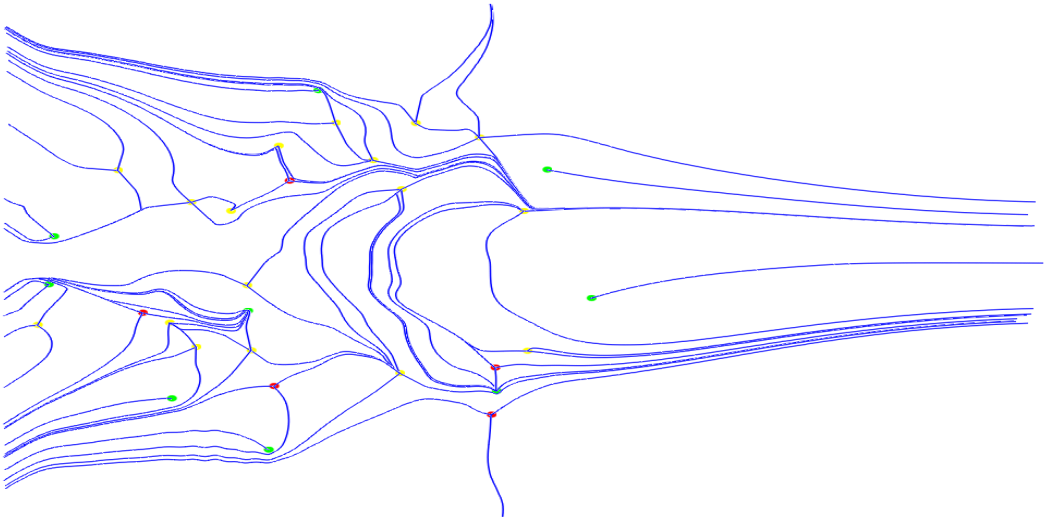


Figure 3: Simplified topology: threshold=0.4, 31 degenerate points

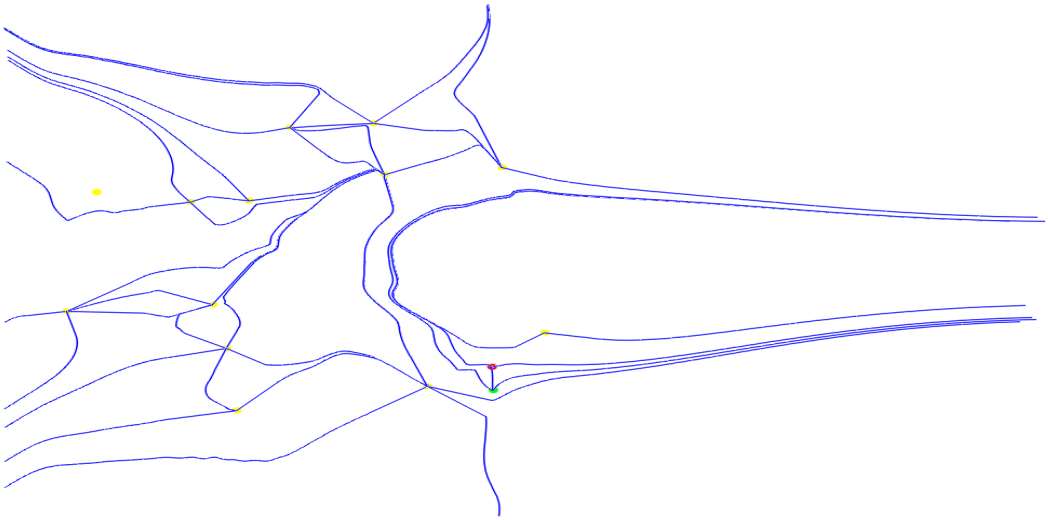


Figure 4: Simplified topology: threshold=0.8, 15 degenerate points

Simplification of Tetrahedral Meshes Using a Quadratic Error Metric

Issac J. Trotts * Bernd Hamann † Kenneth I. Joy†
David Kenwright*

In order to effectively visualize the results of huge numerical simulations it is critical that the size of datasets be efficiently reduced to a manageable size. Various methods have been developed to address this problem by constructing sequences of tetrahedral meshes approximating scalar-valued functions. Several methods rely on iterative vertex insertion. For example, methods described by Hamann and Chen [HC94] and Cignoni *et al.* [CdFM+94] start with a coarse initial tetrahedral mesh and then refine this mesh by inserting vertices in regions having large approximation errors. Such methods provide explicit bounds on the maximal errors at the vertices of the resulting tetrahedral approximations.

Recently, Popovic and Hoppe [PH97] have extended the edge collapse algorithm of Hoppe [Hop96], to arbitrary simplicial complexes. The special case of tetrahedral meshes has been considered by Cignoni *et al.* [CMPS97], Staadt and Gross [SG98], and Trotts *et al.* [THJ99]. The algorithm discussed in [CMPS97] orders edge collapses by considering the variation of the field gradient along edges. The method described by Staadt and Gross orders edge collapses by assigning to each edge a cost that depends on the variation of the scalar value along the edge, the change in volume introduced by collapsing the edge, and the increase in the average deviation from equilateral shape of the tetrahedra affected by the collapse. The methods described by Trotts *et al.* determine an order for edge collapses by computing upper bounds for the function value and domain boundary errors.

*Fluid Dynamics Research Laboratory, Massachussetts Institute of Technology, 37-442 Massachussetts Ave., Cambridge, MA 02319, {trotts,kenwright}@mit.edu

†Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, 1 Shields Avenue, Davis, CA 95616-8562, {hamann,joy}@cs.ucdavis.edu.

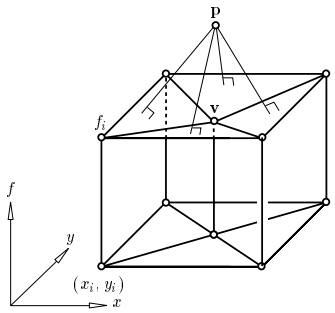


Figure 1: Height field analogy for computation of the error metric. The error of a point $\mathbf{p}$ approximating a vertex $\mathbf{v}$ is given by summing the squared distances of $\mathbf{p}$ to the planes containing the triangles incident at $\mathbf{v}$.

We present a technique which extends the algorithm of Garland and Heckbert [GH98], originally designed to simplify triangle meshes by iteratively merging the vertex pairs leading to least estimated error according to a quadratic (or "quadric") error metric. Garland and Heckbert's method is one of the most efficient triangle mesh simplification algorithms to date [LT99], and computes locally optimal locations for the edge collapse targets. Extending this technique to tetrahedral meshes with associated scalar fields, we use a quadratic error metric that incorporates the shortest perpendicular distance from a given point (the target of an edge collapse) to the hyperplanes spanned by tetrahedra in the region around the collapsing edge. In contrast to the method described earlier by Trotts *et al.* , we measure error in terms of Euclidean distance in 4D space, rather than in terms of a combination of domain boundary error and function value error (see Fig. 1).

In addition, we introduce a spring term to the error metric to keep vertices near their initial locations in 4D. The resulting tetrahedra tend to be more nearly equilateral, provided that this is also true of the tetrahedra in the initial mesh. For datasets with constant or linearly varying regions, such as the *Heaviside3D* dataset shown in Fig. 2, collapses could otherwise happen in an order which would lead to tetrahedra having great variation in edge length.

References

[CdFM+94] Paolo Cignoni, Leila de Floriani, Claudio Montani, Enrico Puppo, and Robert Scopigno. Multiresolution modeling and visualization

Figure 2: Data sets simplified to 1% with non-optimal vertex placement: (a) The Blunt Fin (from 428,202 tetrahedra in 18 minutes on a Pentium III 450 MHZ w/ 256 MB RAM) (b) the Skull (from 2,713,500 tetrahedra in 1 hour, 45 minutes on an R10K with 2GB RAM), (c) Heaviside3D (from 20,250 tetrahedra on an R10K with 512 MB RAM). The Blunt Fin is visualized by several isosurfaces, and the Skull by a single isosurface. The Heaviside3D data set is displayed as its boundary, shaded by scalar value.

of volume data based on simplicial complexes. pages 19–26. ACM, 1994.

[CMPS97] Paolo Cignoni, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, October-December 1997.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In David S. Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization 1998 Proceedings*, pages 263–269, Los Alamitos, California, October 1998. IEEE Computer Society Press.

[HC94] Bernd Hamann and Jiann-Liang Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11:477–489, 1994.

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108, August 1996.

[LT99] Peter Lindstrom and Greg Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98–115, April-June 1999.

[PH97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In Turner Whitted, editor, *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 217–224, August 1997.

[SG98] Oliver G. Staadt and Markus H. Gross. Progressive tetrahedralizations. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization 1998 Proceedings*, pages 397–402, Los Alamitos, California, October 1998. IEEE Computer Society Press.

[THJ99] Issac J. Trotts, Bernd Hamann, and Kenneth I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3), July-September 1999.

Extraction of Crack-free Isosurfaces from Adaptive Mesh Refinement Data

Gunther H Weber[1,2,3], Oliver Kreylos[1,3], Terry J Ligocki[3], John M Shalf[3,4], Hans Hagen[2] and Bernd Hamann[1,3]

[1] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562, USA, {weber, kreylos, hamann}@cs.ucdavis.edu
[2] AG Graphische Datenverarbeitung und Computergeometrie, Fachbereich Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, Germany, {weber, hagen}@informatik.uni-kl.de
[3] Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center, One Cyclotron Road Mailstop 50F, Berkeley, CA 94720, USA, {GHWeber, OKreylos, TJLigocki, JShalf}@lbl.gov
[4] National Center for Supercomputing Applications, 605 E Springfield Avenue, Champaign, IL 61820, USA, jshalf@ncsa.uiuc.edu

1 Introduction

Physical phenomena often vary substantially in scale. There can be large regions where a given physical quantity only varies slightly. At the same time, there can be small regions where the same quantity changes rapidly. Adaptive mesh refinement (AMR) ,see [1, 2, 3], is a technique used in computational fluid dynamics (CFD) to simulate phenomena characterized by drastically varying scales. By using a set of nested grids at different resolutions, AMR combines the topological simplicity of structured rectilinear grids permitting the efficient computation and storage of the result with the adaptivity to changes in spatial resolution of unstructured grids.

The data sets we visualize result from the simulation of astrophysical phenomena by Bryan *et al.* [3] using the Berger and Colella [1] AMR algorithm. They consist of a hierarchy of grids of increasing resolutions. Each level consists of a number of axis-aligned rectilinear grids with data samples associated with the cell centers. Each level has an integer refinement ratio with respect to the parent level, *i.e.*, the cells of the parent level are partitioned in an integer number of refined cells in each dimension. For each grid, except those at the root level, information is provided as to which grid cells in the coarser level they refine.

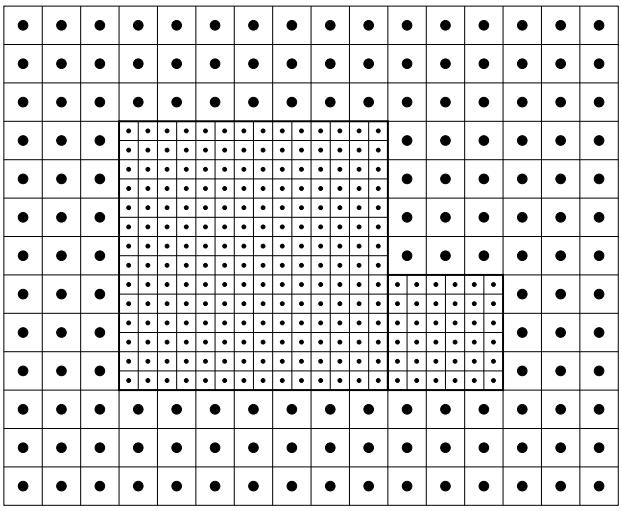


Figure 1: An AMR hierarchy with three grids in two levels.

Figure 1 shows a simple AMR hierarchy consisting of three grids in two levels. The root level consists of one grid. This *coarse grid* is refined by two *fine grids*. The boundaries of the fine grids are drawn as bold lines to make them distinguishable. Locations of the data values are denoted by solid circles.

Isosurface extraction is one of the most commonly used methods for the visualization of scalar-valued volume data. An isosurface is the surface that passes through all points in the volume where the scalar value is equal to a specified *isovalue*. Scalar AMR data poses challenges when one extracts isosurfaces. If close attention is not paid at the borders between different hierarchy levels, discontinuities in the final isosurface triangulation will arise. The goal of our work is to devise a method that avoids these cracks.

2 Related Work

Little research has been published regarding the visualization of AMR data. Norman *et al.* [9] translate an AMR hierarchy into finite element hexahedral cells with cell-centered data and other formats which can take advantage of standard visualization tools (AVS, IDL, and VTK) but preserving the hierarchical nature of the data. Ma [6] describes parallel rendering of AMR data. While he resamples the data to vertex-centered data, he still uses the hierarchical nature of AMR data and contrasts it to resampling it to the finest available resolution. Max [7] describes a sorting scheme for cells for volume rendering and uses AMR data sets as one application of his method.

Isosurfaces are a commonly used technique for the visualization of scalar fields. Our work is based on the marching cubes (MC) method, introduced by Lorensen *et al.* [5]. The volume is traversed cell-by-cell, and the part of the isosurface within the current cell is constructed. This is done by classifying each vertex as either being inside (*i.e.*, its value being less then the isovalue) or outside (*i.e.*, its value being greater than or equal to the isovalue) the isosurface. For each of the 256 possible combinations, there exists an entry in a look-up table (LUT) containing a triangle list for that case. The vertices of these triangles are the intersection points between the isosurface and the edges of the cell. These intersection points are computed by linear interpolation.

The LUT of the original article contained errors which could result in cracks in the extracted isosurface. This is due to ambiguous cases where different possibilities for the isosurface within a cell exist, see Nielson *et al.* [8]. Van Gelder *et al.* [4] provide a detailed description of this problem and describe a solution that produces topologically correct isosurfaces and provide a proof that in order to do so more than one cell must be considered at once.

If topological correctness of the isosurface is sacrificed, it is

possible to take special care in the generation of the LUT, and to avoid the cracks in the final isosurface without looking at surrounding cells. In our work, we use the LUT from VTK, see `http://www.kitware.com/` for further details, which avoids cracks by a carefully devised LUT.

3 First Approach

If the original grid cells for each grid of the AMR hierarchy are used for cell-based isosurface extraction, several problems arise. The MC method expects data values to be associated with cell vertices. The AMR method provides function values at cell centers. Thus, the data set needs to be re-sampled, *i.e.*, interpolated values have to be calculated at cell vertices. This is a problem since AMR simulations are based on the finite-difference method. There is no inherent interpolation scheme for the data that is continuous at cell borders. The MC algorithm in turn approximates the isosurface by interpolating between values at the vertices. The resulting isosurface would be derived from the original data by two consecutive interpolation steps.

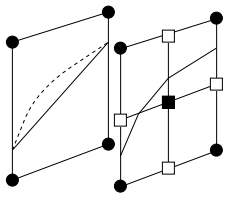


Figure 2: "Dangling node problem" with MC method.

Furthermore, this scheme results in cracks at the borders between two different hierarchy levels. This problem arises, because the resampling yields "dangling" nodes in the fine grid. Even if an interpolation scheme is used that is consistent between levels, *i.e.*, it assigns the same value to the dangling nodes as the interpolation in the coarse grid does, an MC approach still produces cracks in the isosurface.

This is illustrated in Figure 2. The intersection of the isosurface with the cell face is shown as dashed line. The MC method approximates this with a line segment in the coarse grid. For a refinement ratio of two, this coarse cell face is shared with four fine cell faces. The data values present in both grids are shown as solid circles. Additional samples in the fine grid are shown as rectangles. If the values in the outlined rectangles are chosen to be consistent with the values in the coarse grid, there is no problem along the edges because MC will choose the same intersection points for both the coarse and the fine cells. The value in the middle between the four fine cell faces poses a problem. For the isosurface segments to match up, this value must be chosen in a way that the polyline becomes a line. To achieve this, all points on the fine faces would have to lie on a plane (treating the value associated with each point as height). Since this is not generally the case for the four vertices, the point in the middle cannot be chosen in a consistent way with all four vertices.

Other multiresolution methods have encountered the same problem [11, 10] and proposed solutions for it. However, since the "dangling node" problem is a result of the resampling process that in itself poses a few problems, we decided to develop a different approach.

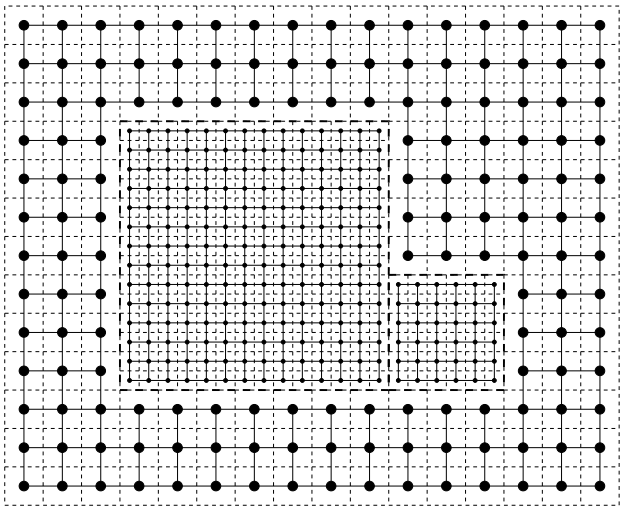


Figure 3: Three AMR grids and their dual grids.

4 Use of Dual Grids

We solve the problems described in the last section by using a *dual grid* for contouring. This dual grid is defined by the original samples at the cell centers by considering them as vertices of a vertex-centered grid. This is illustrated in Figure 3 for the AMR hierarchy from Figure 1. The original AMR grids from Figure 1 are drawn with dashed lines. We note that this process shrinks all grids by one cell in each direction. The result is a gap between the coarse grid and an embedded fine grid. The next section describes how this gap can be used to avoid discontinuities in the final isosurface. The dual grid approach generalizes to the 3D case.

5 Stitching 2D Grids

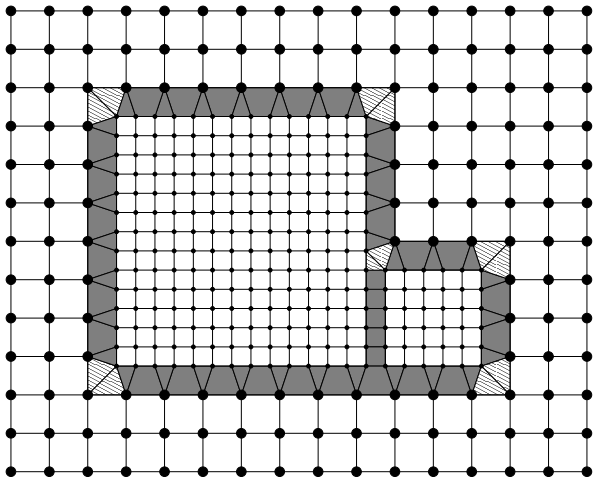


Figure 4: "Stitch cells" between the three grids shown in Figure 3.

The use of the dual grids results in a gap between the hierarchy levels. This gap can be used to merge the different hierarchy level seamlessly. We fill this gap in a *stitching step*. The resulting *stitch mesh* is constrained by the boundaries of the coarse and the fine grids. Furthermore, it must not subdivide any edges of the existing

grids. In 2D, this is achieved by requiring that only existing vertices are used and no new vertices generated. Since one of the reasons for using the dual grids is avoiding the insertion of new vertices, whenever possible, this poses no problems.

In the 2D case, a constrained Delaunay triangulation can be used to fill the gap between grids. For two reasons, we chose not to do this. First, while in the 2D case only edges must be shared between the stitching grid and the dual grids, in the 3D case, faces must be shared. The boundary faces of the rectilinear grids are quadrilaterals and cannot be shared with tetrahedra without being subdivided, causing cracks for the reasons explained in Section 3. Furthermore, an index based approach is faster, since it uses the regular structure of the boundaries while avoiding problems that might be caused by this regular structure when using a Delaunay-based approach.

The stitching process for a refinement ratio of two is shown in Figure 4. Stitch cells must be generated for edges along the boundary and for the vertices of the fine grid. The stitch cells generated for the edges are shown in grey, while the stitch cells generated for the vertices are drawn crosshatched. For the transition between a fine and a coarse grid, each edge of the fine grid is connected alternatingly to either a vertex or edge of the coarse grid. This yields triangles and deformed quadrilaterals as cells. The quadrilaterals are not subdivided, since such a subdivision is not unique. This in turn would again result in problems in the 3D case when these quadrilaterals are shared between cells.

In the case of multiple grids, a check must be performed: Are the grid points in the coarse grid refined or not? Figure 4 shows several cells that connect to an adjacent fine grid.

The vertices are connected to the coarse grid via two triangles. Here, a consistent partition of the deformed quadrilateral is possible. Again, this partition was chosen to avoid problems in the 3D case. We note that, in the vertex case, it is also important to consider whether adjacent cells are refined. Potentially, if another fine grid is adjoining, an edge must be created between the current grid and the adjacent grid. This edge is then connected to the coarse grid. This case is illustrated along the bottom edge of the fine grids shown in Figure 4.

6 Stitching 3D Grids

The index-based approach can be generalized to the 3D case. In the simple case of one fine grid embedded in a coarse grid, quadrilaterals, edges and vertices of the fine grid must be connected to the coarse grid. Along each of the two directions of the face, a decision is made to connect to a vertex or an edge. The combination of these two decisions results in each quadrilateral being connected to either a vertex, a line segment (in the two possible directions) or another quadrilateral. The cell types resulting from these connections are pyramids, deformed triangle prisms and deformed hexahedral cells, shown in Figure 5.

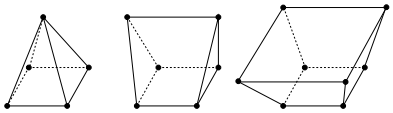


Figure 5: Cell types resulting from connecting a fine grid face.

The edge case can be thought of as a combination between vertex and edge case of the 2D case. If the viewing direction is parallel to the edge (so that it appears to the viewer as a point), it must always connected to two perpendicular edges of the coarse grid. In the direction along the edge, one connects it to a point or a parallel edge. The combination results in the edge to be connected to either two perpendicular edges or two quadrilaterals of the coarse grid. This results either in tetrahedra or deformed triangle prisms as connecting cells, shown in Figure 6.

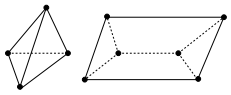


Figure 6: Cell types resulting from connecting a fine grid edge.

The vertex case is the combination of two 2D vertex cases. This results in each vertex being connected to three quadrilaterals of the coarse grid via pyramid cells.

In the case of the coarse grid being refined by more than one fine grid, each coarse grid point must be checked for being refined. Edges might be “upgraded“ to the quadrilateral case (for two adjacent edges). Vertices can be “promoted” to edges or even quadrilaterals, for the case of more than two grids meeting at a given location. This results in a large number of possible cases to be considered.

7 Results

The results of using dual grids, generating stitch cells and extending MC to handle the new polyhedral cell types is shown in Figure 7. It shows an isosurface created form two levels of an AMR hierarchy. One fine grid is embedded in a coarse grid. The isosurface generated for the coarse grid is colored red, the isosurface generated for the fine grid is colored blue, and the isosurface generated for the stitch cells is colored green. The resulting isosurface is seamless at the boundaries between the two levels.

8 Future Work

Future work will be directed at multiple grids embedded in a coarse grid. Furthermore, the use of a generic triangulation scheme satisfying the outlined constraints is under consideration. This would allow the use of our method for other AMR data, where the grids are not necessarily axis-aligned, *e.g.*, data sets produced by the AMR method of Berger *et al.* [2].

9 Acknowledgments

This work was supported by the Directory, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098, the National Science Foundation under contracts ACI 9624034 and ACI 9983641 (CAREER Awards), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of ALSTOM Schilling Robotics, Chevron, General Atomics, Silicon Graphics, and ST Microelectronics, Inc. We thank the members of the NERSC/LBNL Visualization Group; the LBNL Applied Numerical Algorithms Group; the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis and the AG Graphische Datenverarbeitung und Computergeometrie at the University of Kaiserslautern, Germany.

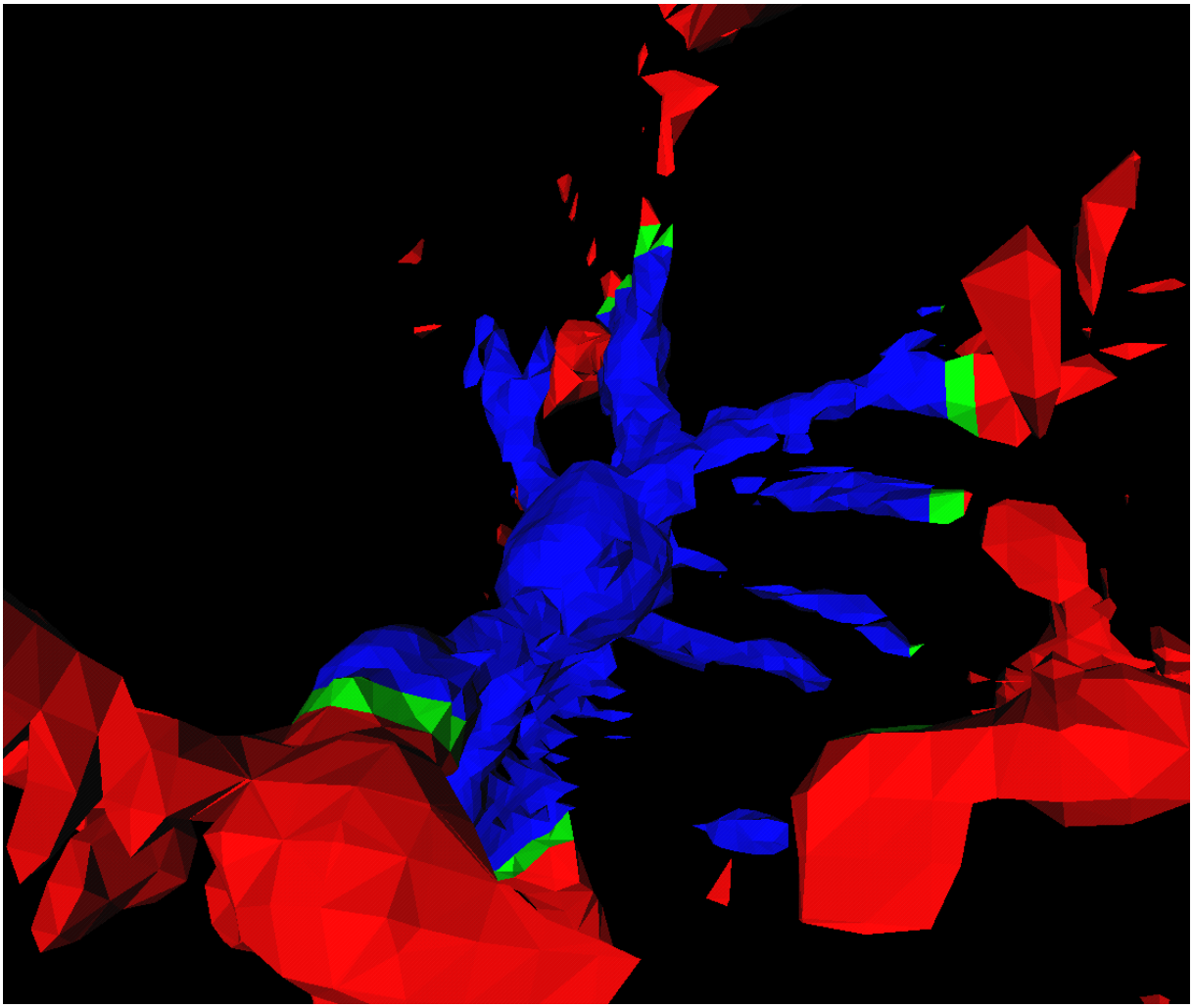


Figure 7: Isosurface extracted from two levels of the AMR hierarchy (data set courtesy of Greg Bryan, MIT, Theoretical Cosmology Group).

References

[1] Marsha Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989. Lawrence Livermore Laboratory Report No. UCRL-97196.

[2] Marsha Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, March 1984.

[3] Greg L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, March/April 1999.

[4] Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994.

[5] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[6] Kwan-Liu Ma. Parallel rendering of 3d amr data on the sgi/cray t3e. In *Proceedings of Frontiers '99 the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 138–145. IEEE Computer Society, February 1999.

[7] Nelson L. Max. Sorting for polyhedron compositing. In H. Hagen, H. Mueller, and G. Nielsen, editors, *Focus on Scientific Visualization*, pages 259–268. Springer-Verlag, 1993.

[8] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91, 1991.

[9] Michael L. Norman, John Shalf, Stuart Levy, and Greg Daues. Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Engineering*, 1(4):36–47, July/August 1999.

[10] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surface. In *Proceedings of the 7th Annual IEEE Conference on Visualization (VIS-96)*, pages 335–342, 499, New York, October 1998. ACM Press.

[11] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, 1999.

Title: Hierarchical Best Linear Spline Approximation

Speaker: David F. Wiley
Department of Computer Science
University of California
One Shields Avenue
Davis, CA 95616
U.S.A.

Email: wiley@cs.ucdavis.edu
Phone: (530) 754-9470

Authors: David F. Wiley, Martin Bertram, Benjamin W. Jordan, and Bernd Hamann

Abstract:
We present a method for the hierarchical approximation of functions in one, two, and three variables. The input to our method is a coarse decomposition of the compact domain of a function in the form of intervals (univariate case), triangles (bivariate case), and tetrahedra (trivariate case). We compute best linear spline approximations, understood in an integral least squares sense, for functions defined over triangulations and refine triangulations using repeated bisection. This requires the identification of the interval (triangle, tetrahedron) with largest error and splitting it into two intervals (triangles, tetrahedra). Each bisection step requires the recomputation of all spline coefficients due to the global nature of the best approximation problem. This is done efficiently by bisecting multiple intervals (triangles, tetrahedra) in one step and by using sparse matrix representations for the matrices resulting from the normal equations. The spline coefficients are readily found using an efficient sparse matrix system solver.

The linear spline we use for the approximation function is a linear combination of hat basis functions with associated spline coefficients. At an arbitrary spline knot location the associated hat function has a value of 1 and varies linearly to 0 from this knot to the neighboring knot(s) and is 0 at all other knot locations. From approximation theory, the spline coefficients can be found using the system $Mc = F$ where M is an NxN matrix (where N is the number of spline knots) whose entries are the inner products of the hat functions; c is the spline coefficient vector of size N; and F is a vector of size N whose entries are the inner products of the hat functions with the input function.

A spline segment in the univariate case and a minimal triangulation of the convex hull of the input function in the bivariate and trivariate case initially approximate the input function. Spline coefficients are computed for the initial approximation to produce the first level of the hierarchy. Error between the approximation function and input function is estimated using the L^2 norm of the difference between the two functions. Error estimation is performed at an interval (triangle, tetrahedron) level such that local errors are established. The top 10% of intervals (triangles, tetrahedra) with the highest local error values are bisected and spline coefficients recomputed to produce the next

successive level in the hierarchy. These steps are repeated until a global error tolerance is achieved.

Due to the nature in which we refine the approximation at each iteration, we can take advantage of the small changes that occur in the system Mc = F between each level. Unaffected entries are reused and the entries that are affected are computed. It is possible to implement a system solver that takes advantage of these small changes as well, although in our particular implementation we found that much more computation time is spent updating F than the combined time of updating M and using a conventional sparse solver to find the coefficients. The cause of this bottleneck is the need for high-resolution integration over the input function because low-resolution integration schemes result in poor spline coefficients that are not useful. Our implementation relies upon a Romberg integration scheme that produces good results.

When approximating a function containing finite data, an improvement on the method is to consider only original data sites during the bisection step. Here, the splitting of the interval (triangle, tetrahedron) occurs at the nearest original data site to the midpoint, not at the analytical midpoint as before. This provides a natural upper bound to the total number of subdivision steps and permits the reference of original data sites. Approximation storage requirements are reduced as a result.

First derivative information can be used in the spline coefficient computation to refine the approximation. Using this information reduces over- and under-shoots near high gradient areas that appear in the spline approximation. The amount of first derivative information use is controlled by weight coefficients to allow more flexibility in the approximation.

First derivative information can also be used in the error computations to penalize intervals (triangles, tetrahedra) that cover high gradient regions. These intervals (triangles, tetrahedra) are then more likely to be split during the bisection step. The result is a higher concentration of intervals (triangles, tetrahedra) near high gradient regions and fewer in the less interesting low gradient field. Again, the contribution of the first derivative component is governed by weight coefficients.

Modify the inner product used above in the spline coefficient and error computation to include the weighted first derivative information. The applicability of the first derivative information depends upon the input function. It is difficult to choose a single set of weights to govern the first derivative information for all types of input.

Goals considered in the design of this method are: simplicity, generality, efficiency, compact representation, and applicability. Simplicity of the method is guaranteed by the use of well-known spline approximation methods and small topological changes during subdivision. The method is general enough to be applied to functions of any number of variables as well as multi-valued functions. Computation is reasonably efficient when using sparse matrix representations. The generated approximations can be stored compactly since only the coefficients and small topological changes occur between each

level, and the approximations are easily rendered using conventional visualization techniques.