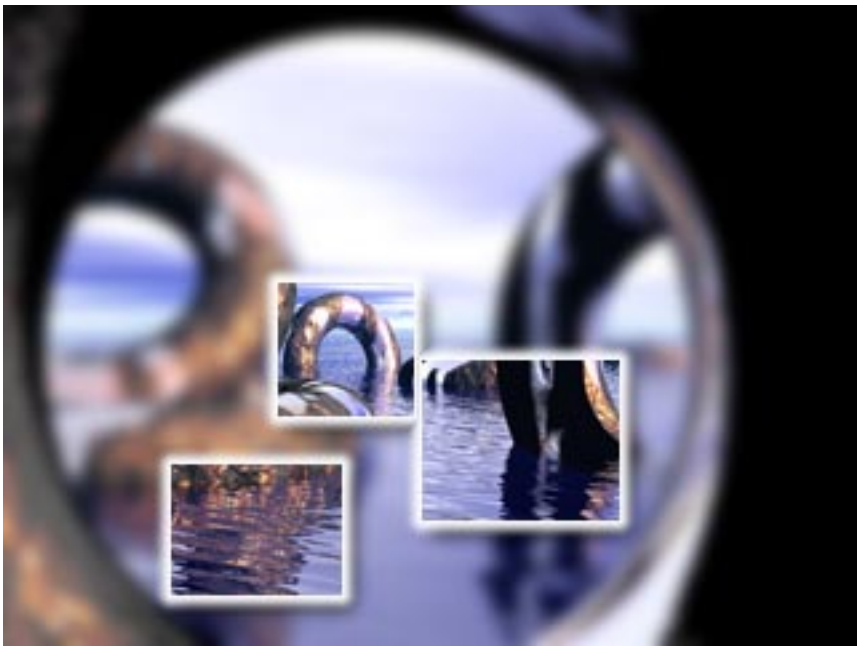
Crossroads
The ACM Student Magazine

An Error-Controlled Octree Data Structure for Large-Scale Visualization



by ***Dmitriy V. Pinskiy***, ***Joerg Meyer***, ***Bernd Hamann***, ***Kenneth I. Joy***,***Eric Brugger***, and ***Mark A. Duchaineau***

Introduction

Motivation

The fields of medical imaging, vector field visualization, flow simulation, and computational fluid dynamics (CFD) produce large data sets. These datasets cannot be rendered in a reasonable amount of time. Rendering a complete data set at high resolution is often time-consuming and intractable. Moreover, if a user is interested in only one small portion of a data set, it is wasteful to render the whole data set at high resolution. For example, a medical researcher might be interested not in an entire MRI data set, but only in a certain subregion.

Figure 1. Original data set.

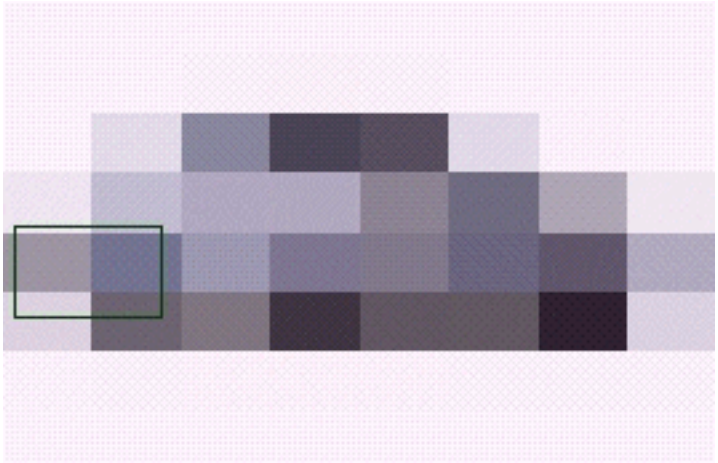


Figure 2. Main region representations.

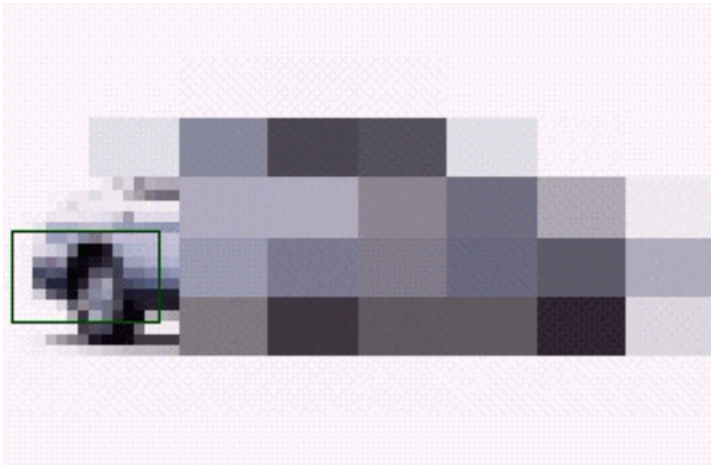


Figure 3. Main and subregion representations.

Multiresolution Approach

One might solve the above problem by viewing different regions, of a dataset, at different resolutions. Before isolating specific regions to render at higher resolutions, a user must first specify the main region. The **main region** is the portion containing the region of interest (ROI). The main region can be either the entire data set or a subset of it. Our algorithm will then generate a low resolution octree representation of the main region. **Octrees** are similar to binary trees. However, an octree has eight children on each internal node. In our representation, each node corresponds to a cubic region. The octree provides a natural mechanism for space subdivision. We render the main region at a low resolution to accelerate the visualization as shown by Figures 1 and 2.

While a low resolution rendering does not provide great detail, it is often sufficient approximation for a user to locate the ROI. Only the ROI, usually a small subregion of the whole data set, should be rendered at the highest resolution as shown in Figure 3.

Related Work

Our algorithm subdivides the octree representation of space when approximation error exceeds a threshold or when a user requests a refinement to a ROI. We begin with an initial, coarse, octree decomposition of a three-dimensional (3D) bounding box of the domain of a given volumetric data set. Initially, octree cells are split merely based on a local error estimate. The splitting process is terminated when a maximum number of cells is exceeded or when all cell-specific errors are smaller than some specified error threshold.

We use the difference between the original data values and the constant value approximations, within each octree cell, as an error metric. The term **data-dependent discretization** refers to decompositions of space that approximate a continuous function by a piecewise constant or linear function. Previous work on data-dependent discretization techniques is covered in [**5**], [**6**], and [**14**].

Using an octree-based approximation, one generates a data-dependent, spatial decomposition that adapts to the underlying complexity of the input data. The techniques described in [**8**], [**9**], and [**10**] deal with the problem of decimating triangular surface meshes and adaptive refinement of tetrahedral volume meshes, respectively. These two approaches are aimed at the concentration of points in regions of high curvatures (or high second derivatives). This principle can be used to either eliminate points in nearly linearly varying regions (decimation) or to insert points in

highly curved regions (refinement). The data-dependent, octree-based, decomposition scheme we describe in this paper is based on the principle of refinement: Our algorithm inserts octree cells when the error is large or when a user requests local refinement.

Our technique constructs a data pyramid. A **data-pyramid** is a data hierarchy of cells with increasing precision [**3**]. The pyramid concept has also been extended to the adaptive construction of tetrahedral meshes for scattered scalar-valued data [**1**][**2**]. So-called multiresolution methods have been developed for polygonal and polyhedral approximations of surfaces, graphs of bivariate functions, and scalar fields defined over volumetric domains. Such approaches are described in [**4**], [**7**], and [**11**]. Our data-dependent, octree method can be viewed as a combined automatic and user-driven, hierarchical scheme supporting the visualization of large-scale, scientific data by multiple levels of approximation. Some of the fundamental concepts from computational geometry we are using are discussed in depth in [**13**].

Data Structure Issues

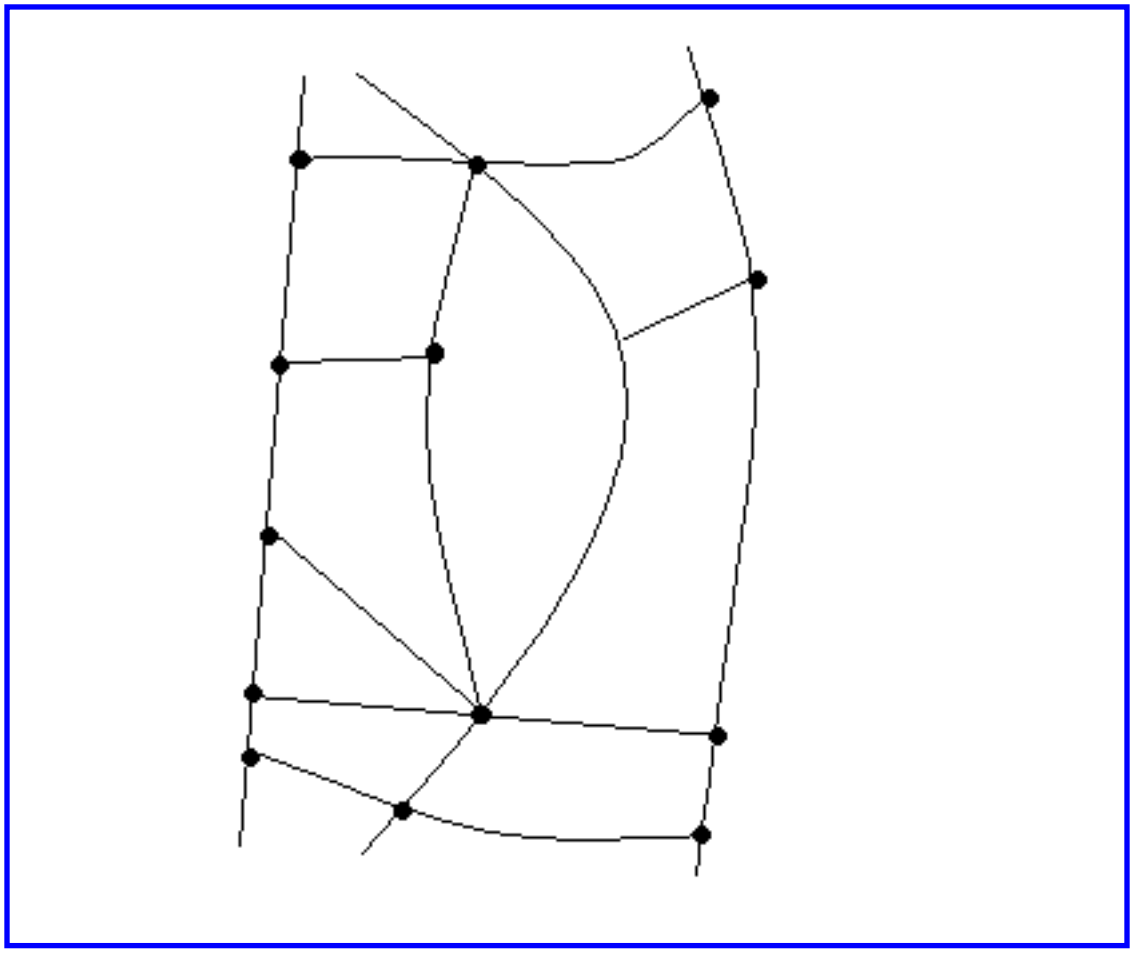


Figure 4. Example of an irregular grid.

Points

Suppose our data set is determined on an arbitrary, irregular grid without interior holes as shown in Figure 4. Each mesh element (cell) has an associated data value such as color, density, weight, temperature, etc.

One of the most fundamental data structures used in our algorithm is a set of points. Each point corresponds to a cell of the original mesh. Each point describes an aspect of a mesh element such as:

- coordinates defining the center
- associated data values
- area (2D case) or volume (3D case) of the mesh element

In practice, it is convenient to maintain the set of points as a dynamically allocated array, and to use point indices rather than actual point coordinates.

Resolutions and Representations of Main and Subregions

We store two numbers defining the resolutions of the main region and the subregion. We also store two pointers to represent the main and the subregion. Each region is represented by a quadtree or octree data structure in the 2D and 3D case, respectively.

Quadtree and Octree as a Representation of a Region

Each leaf, in a tree, corresponds to a different subregion. The union of these subregions covers the entire region. We display the region by drawing and coloring the bounding boxes of each leaf with average function value *F* calculated by the formula:

$$F = \sum_{i=1}^{N} \frac{a_i * f_i}{A}$$

where *A* is the area or volume of the bounding box of the node,

ai is the area or volume associated with the i-th point, and

fi is the function value associated with the i-th point.

Error

We assume that a mesh cell belongs to a quadrant or octant if the center of the mesh cell is inside the bounding box. We can safely neglect error caused by this assumption as illustrated by the following argument: Large data sets imply a large number of mesh cells. Assume that the area or volume of the whole data set is constant and that the area or volume of a single cell is relatively small. Given these assumptions, it is relatively inconsequential if a cell lies only partially inside, entirely inside or entirely outside a box as the area or volume of the bounding box can be approximated by the sum of the areas/volumes stored as part of the points of the leaf.

We compute two errors for the bounding box, the root mean square (RMS) error and the maximum error (MAX):

$$E_{rms} = \sqrt{\frac{\sum_{i=1}^{N} (f_i - F)^2}{N}}$$

$$E_{\max} = \max(|\ f_i - F\ |)$$

where *F* is the average of all function values associated with the box.

To visualize the error of an approximation, we use gray-scale shading. In Figures 5 and 6 white corresponds to zero error, and black corresponds to the highest error.

Figure 5. RMS errors for main region representations.

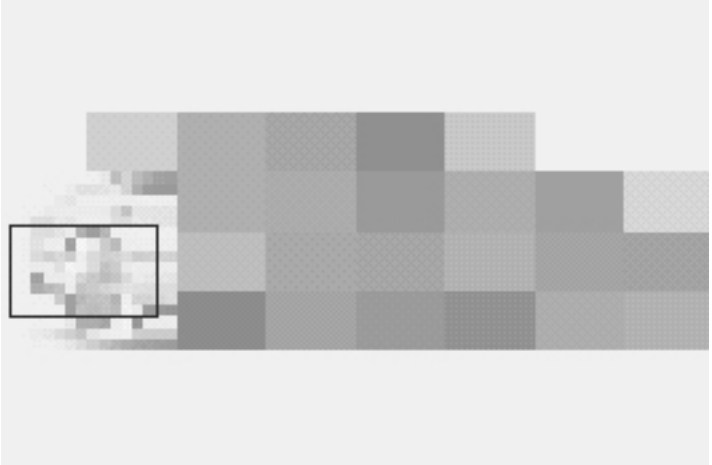


Figure 6. RMS error for main and subregions representations.

To reduce the error, we decrease the number of points per box and make the boxes smaller. A smaller region represented in the octree corresponds to smaller boxes of the octree's leaves. Thus, when a user specifies a relatively small region in the original volume, a high detail image results. To improve the quality of the resulting image without decreasing the area of the region, we need to increase the depth of the octree. Increasing the number of subdivisions (constructing a deeper tree) corresponds to increasing the resolution in the image. If a region is small enough to contain at most one point it will theoretically have zero error.

At first blush, the error calculation appears to be a wasteful and time-consuming operation. However, in practice, the error function can accelerate the whole algorithm. In near constant-value regions, the error will be smaller than a tolerance *e*. In these regions, the number of subdivisions needed is significantly less than the maximally allowed depth of the tree.

Applications

We initialize the set of points, in our data structure, from an input data set and build the main octree corresponding to the main region with a depth. Typically, the depth of this main tree is low, as the main region need not be represented in great detail. Initializing the set of points and computing the main octree is straightforward as shown in Figure 7. After constructing the main tree, we can display the main region at low resolution, and the user can select a ROI.

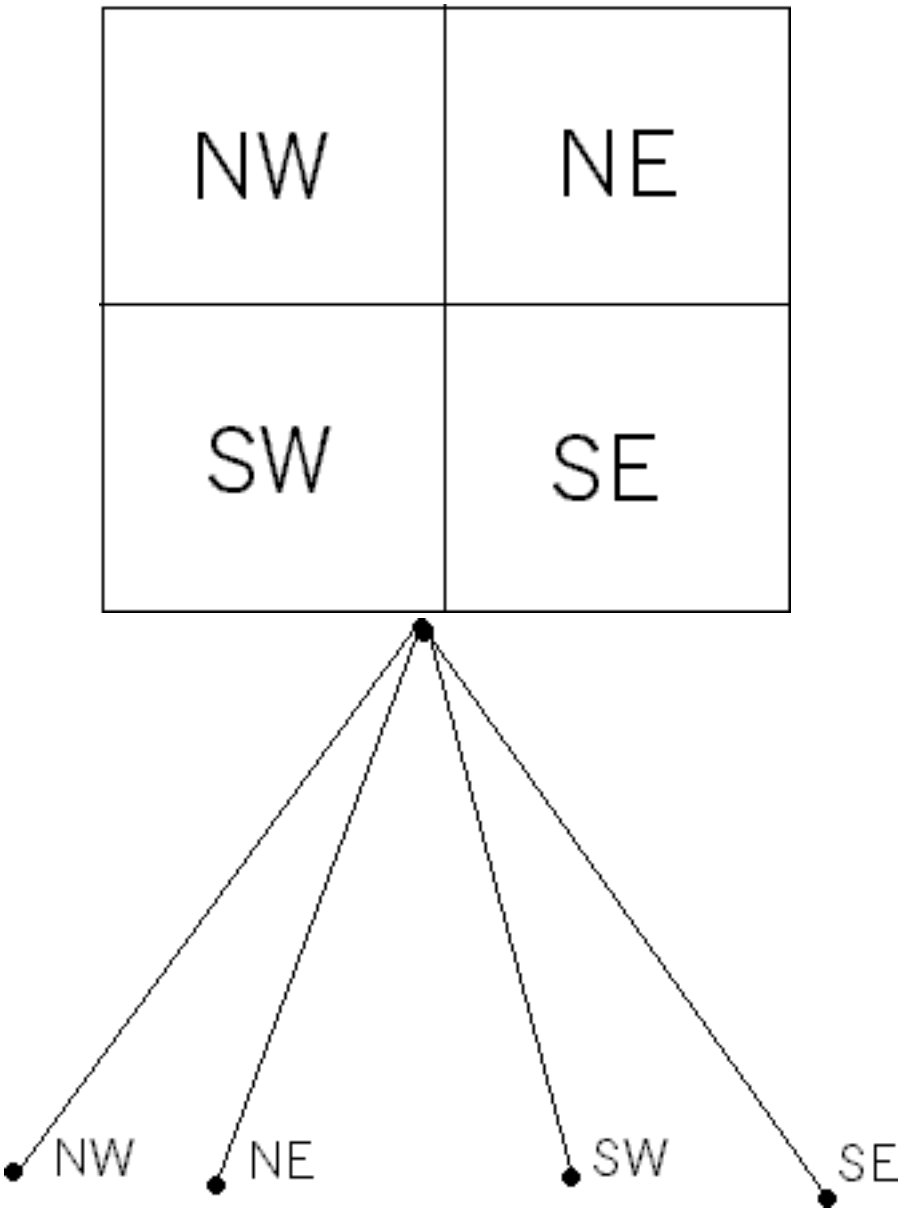


Figure 7. Main region and corresponding quadtree (2D case).

We next modify modify our tree to construct a tree with greater depth in a selected subregion. To do this efficiently, we take advantage of the fact that the points are already sorted in the main tree. We use the main tree as a search tree and can easily find a node *A* whose bounding box contains our subregion. When building the suboctree, we do not consider all points. Instead we only consider those points inside the bounding box of node *A*. We therefore process a smaller number of points and speed up the algorithm. Figure 8 shows the selection of a box containing a desired subregion.

Figure 8. Subregion inside main region not including center of main region. (Note: To build an octree for a subregion we consider only points of the SW branch of the main tree root, and we do not consider the portion of the textured points.)

Figure 9. Subregion inside main region including center of main region. (Note: Space was subdivided at an uneven ratio, and, therefore, to build an octree for the sub region, we consider only points of the SW branch of the main tree root and do not consider the points the within textured boxes.)

Figure 10. Subregion inside the main region including center of main region. (Note: To build an octree for a subregion we consider only points of the NW-SE, NE-SW, SW-NE, and SE-NW branches of the main tree root and we do not consider points within the textured boxes.)

However, if the subregion is near the center, the node whose bounding box contains the whole subregion might be the root making it necessary to re-process all points to build the sub-tree. There are several options to deal with this problem. If we knew the central location of the ROI *a priori* then we could subdivide the space into uneven segments (for example 1/3 or 2/3) when constructing the main tree as shown in Figure 9. Alternately, if node *A* is root, we can traverse the path from node *A* towards the leaves while purging points outside the subregion. Purging extraneous points is accomplished by ignoring all branches and leaves whose bounding boxes do not touch the ROI as shown in Figure 10.

Implementation and Results

Tree data structure

Each node of the tree consists of the following fields:

- a pointer to the parent node;
- flags to indicate whether a node is a nonempty leaf, an empty leaf, or a branch; and
- coordinates of the bounding box that contains all points associated with the node.

The root's bounding box includes all points while leaves have the smallest bounding boxes in the tree.

When a node is a branch, it also has fields such as a pointer to the North-West front node, a pointer to the South-East rear node, etc. In case of a non-empty leaf, the node includes the set of points (actually a set of indices to points) that belong to the leaf. In case of an empty leaf, it is sufficient for the node to have only the three basic fields. To visualize an octree, we can simply draw wire frames of bounding boxes of the octree nodes.

Calculating Errors

Consider the data set shown in Figure 1; a slice through a 3D data set. Each point, of this data set, has an associated RGB color value. To calculate the maximum error for an RGB value, we calculate the maximum error for the red, green, and blue color components and then choose the maximum of these three errors.

To calculate the RMS error, we use the following formula:

$$Error = \left(\sum_{i=1}^{N} di \;/\; N\right)$$

where *di* is the distance between a ``point'' (*ri*, *gi*, *bi*) and the ``average point'' (*R*, *G*, *B*), and *N* is the total number of points. The values of *R*, *G*, and *B* are the components of the average color of the leaf; the values of *ri*, *gi*, and *bi* are the components of the color associated with the *i*-th point of the leaf. Thus, the above formula shows how much a RGB value of the *i*-th point varies from the average.

Tree Depth and Error

Increasing the depth of the tree decreases the error. For the data set shown in Figure 1, a main tree depth of two, leads to an average value of the RMS errors of 140 whereas a depth of four leads to an RMS error of 59. Increasing the depth by factor of two decreases the RMS error more than twice.

By altering the subtree depth as shown in Figure 3 one might reduce the RMS error from 132 at a depth of two to 66 at a depth of four. These results seem surprising.

When both trees have the same depth (four), the RMS error of the representation of the small subregion is greater than the error of the representation of the main region. Figure 3 might create the impression that one perceives a subregion in greater detail. We note that the far South and the far North parts of the main region are completely white. As a result, we have an extremely good approximation. In the subregion, we do not have such constant-colored spots, and the RMS error is high. However, the maximum error in the subregion is slightly lower than in the main region. (For example, for a depth of four, we obtain a maximum error of 210 for the subregion representation and 225 for the main region representation.)

Conclusion and Future Work

Our method allows a user to define an ROI in a 2D/3D data set, which will be rendered at the highest resolution, while the rest of the data set, or some part of it, will be rendered at a lower resolution. The method can be used to navigate within the data set, so that the current region of interest is always displayed at a higher resolution than less important regions. Therefore, it provides a flexible interface, which allows arbitrary positioning of the ROI in 2D/3D space [**12**]. The underlying data structure of our algorithm is a tree (quadtree for the 2D case and octree for the 3D case). Setting specific resolution and allowed error tolerance, we prevent the tree structure from growing excessively in depth.

Since the main region would be relatively large and we are using a low resolution to visualize it, we can assume that each leaf contains at least one point. We also assume that the error is greater than the tolerance *e* because otherwise a user should be satisfied with the quality of the main region, and there would be no reason for having a subregion. The additional overhead for allocating and storing the pointers suggests considering a hashing method as an alternative for the main region representation. If we were to use this approach, each square or cube would take less memory than a leaf, since we would not allocate memory for pointers to parents, flags, or a bounding box. Using hashing, we would associate a two-number index with each square or a three-number index with each cube. Knowing the indices and the size of the squares or cubes, we would calculate all necessary information to construct the tree for the subregion. This technique would help to accelerate the algorithm even more and contribute to saving time and memory.

Acknowledgements

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award); 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 Shilling Robotics, Davis, California; Chevron; and Silicon Graphics, Inc. We thank the members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

1

Bertolotto, M., De Floriani, L. and Marzano, P. *Pyramidal simplicial complexes.* Third Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, Hoffmann, C. and Rossignac, J., eds. ACM Press, Association for Computing Machinery, New York, NY., 1995, pp.153-162.

2

Cignoni, P., De Floriani, L., Montani, C., Puppo, E. and Scopigno, R. *Multiresolution modeling and visualization of volume data based on simplicial complexes.* Symposium on Volume Visualization, October 17-18, 1994, Washington, D. C., Kaufman, A. E. and Krueger, W., eds., (1994.) IEEE Computer Society Press, Los Alamitos, CA., 1994 pp. 19-26.

3

De Floriani, L. *A pyramidal data structure for triangle-based surface description.* IEEE Computer Graphics & Applications 9(2). 1989, pp. 67-78.

4

DeRose, A. D., Lounsbery, M., Warren, J. *Multiresolution analysis for surfaces of arbitrary topological shape.* Technical Report 93-10-05, Department of Computer Science and Engineering. University of Washington, Seattle, WA., 1993.

5

Dyn, N., Levin, D., and Rippa, S. *Data dependent triangulations for piecewise linear interpolation.* IMA Journal of Numerical Analysis 10, 1988 pp.137-154.

6

Dyn, N., Levin, D., and Rippa, S. *Algorithms for the construction of data dependent triangulations.* Algorithms for Approximation II. Mason, J.C. and Cox, M.G., eds., Chapman and Hall, New York, NY., 1990, pp.185-192.

7

Eck, M., DeRose, A. D., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W. *Multiresolution analysis of arbitrary meshes.* Proceedings of SIGGRAPH 1995, Cook, R., ed. ACM Press, New York, NY., 1995, pp. 173-182.

8

Hamann, B. *A data reduction scheme for triangulated surfaces.* Computer Aided Geometric Design 11(2), 1994, pp.197-214.

9

Hamann, B. and Chen, J. L. *Data point selection for piecewise linear curve approximation.* Computer Aided Geometric Design 11(3), 1994a, pp. 289-301.

10

Hamann, B. and Chen, J. L. *Data point selection for piecewise trilinear approximation.* Computer Aided Geometric Design 11(5), 1994b pp. 477-489.

11

Lounsbery, M. *Multiresolution analysis for surfaces of arbitrary topological shape, dissertation.* Department of Computer Science and Engineering, University of Washington, Seattle, WA., 1994.

12

Meyer, J., Hagen, H., Lohr, C., Deitmer, J. W. *Interactive Navigation through Glial Cells.* Computer Graphics International '98, Minisymposium on ``Scientific Visualization'', June 22-26, 1999, Hannover, Germany, 1998, pp. 73-77.

13

Preparata, F. P. and Shamos, M. I. *Computational Geometry.* Third printing, Springer-Verlag, New York, NY., 1990.

14

Schumaker, L. L. *Computing optimal triangulations using simulated annealing.* Computer Aided Geometric Design 10(3-4), 1993, pp. 329-345.

Biography

Dmitriy V. Pinskiy is currently an undergraduate student at the Computer Science department at UC Davis. His research focuses on multiresolution techniques and large-scale visualization.

Joerg Meyer is a post-doctoral lecturer and research scholar at UC Davis. He received his Ph. D. from the University of Kaiserslautern, Germany. His research topics include volume visualization, medical and biomedical imaging, and interactive rendering.

Bernd Hamann is an Associate Professor of Computer Science and Co-Director of the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and an Adjunct Professor of Computer Science at Mississippi State University. He collaborates as a Participating Guest with the Lawrence Livermore National Laboratory. Professor Hamann's main research and teaching interests are visualization, geometric modeling/computer-aided geometric design (CAGD), computer graphics, and immersive visualization environments.

Kenneth I. Joy, member of the Association for Computing Machinery (ACM), is a professor in the Computer Science Department at the University of California at Davis. He came to UC Davis in 1980 in the Department of Mathematics and was a founding member of the Computer Science Department in 1983. Professor Joy's research and teaching areas are visualization, geometric modeling, and computer graphics.

Eric Brugger (B.S. computer science and physics, University of California, Berkeley), Lawrence Livermore National Laboratory, is team leader for computer scientists working on Mesh-TV, an interactive graphical analysis tool for visualizing two- and three-dimensional data. His research interests include exploration methods for vector fields on unstructured meshes and massive parallel architectures.

Mark A. Duchaineau, Lawrence Livermore National Laboratory, has been affiliated with Los Alamos National Laboratory, the U.C. Davis Computer Graphics Group (Center for Image Processing and Integrated Computing), and the Department of Mathematics and Computer Science at C.S.U. Hayward. His current research centers primarily on scientific visualization of massive data sets.