A Hierarchical Error Controlled Octree Data Structure for Large-Scale Visualization

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

Abstract

We present an octree-based approach supporting multiresolution visualization of large three-dimensional scientific data sets. Given an irregular gridded data set, we initially impose an octree data structure of relatively low resolution, i.e., consisting of a relatively small number of cells. The construction of this initial octree structure is controlled by the original data resolution and cell-specific error values. For each cell in the initial octree structure, we compute the average field value, and a cell-specific error value. It is thus possible to use the octree to visualize either the field function value or the local error value. The octree data structure can be refined further in areas that are specified by a user of a visualization system: A user would identify a region in space, i.e., an octree cell, where the field is of greater interest or where octree cells carry relatively large error values. This usage of our data structure ensures that we use the highest resolution to render only regions of interest in a large-scale scientific data set.

Introduction

Motivation

Numerous applications in the field of medical imaging, vector field visualization, flow simulation, and computational fluid dynamics (CFD), produce large data sets, which cannot be rendered anymore with today's methods in a reasonable amount of time. Rendering a complete data set at the highest resolution often becomes very time-consuming and unfeasible. Moreover, applying the highest resolution to the whole data set could be wasteful if a user is interested in only one small portion of a data set. For example, a medical researcher might be interested not in an entire MRI data set, but only in a certain subregion.

Multiresolution Approach

A solution to the above problem is to apply different resolutions for different parts of a data set. First, a user specifies the *main region* - a portion that contains the region of interest (ROI). The main region can be either the entire data set or some part of it. Using a *low resolution*, our algorithm generates an octree representation of this main region. Octrees are similar to binary trees, but octrees' internal nodes have eight children, and each node corresponds to a cubic region. Thus an octree represents a method for space subdivision. Even when one has to render a large main region, we render it at a low resolution to accelerate the visualization (Figures 1 and 2).

Certainly, low resolution does not provide a lot of detail, but a preview rendering is often sufficient for a user to locate the specific region of his or her interest inside the main region; and only this specific region, which usually is a small subregion of the whole data set, should be rendered at the highest resolution (Figure 3).

FIGURE 1.
Original data set.

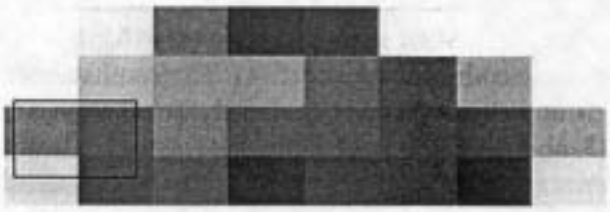

FIGURE 2.
Main region representations.

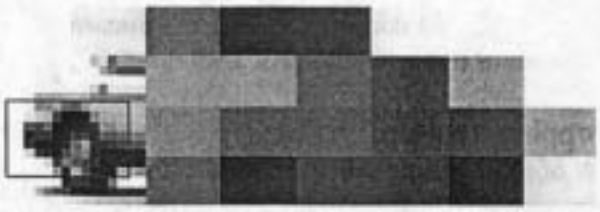

FIGURE 3.
Main and subregion representations.

Related Work

Our algorithm is based on subdividing an octree representation of space whenever the approximation error exceeds a certain threshold or when a user requests a further refinement in a designated *region of interest* (ROI). We start 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 approximation 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.

For an error metric, we use the differences between original data values and the constant values associated with each octree cell. Often, one uses the term "data-dependent discretization" for decompositions of space when the goal is to closely approximate some function by a certain 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 very closely to the underlying "complexity" of the field function defined by 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 that 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.

In principle, our technique is related to the idea of constructing a "data pyramid," i.e., 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

Points

Suppose our data set is determined on an arbitrary irregular grid without holes inside (Figure 6). Each mesh element (cell) has an associated data value that can be, for example, color, density, weight, temperature, etc.

FIGURE 4.
RMS error for main regions.

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 consists of

- coordinates, which define the center of the corresponding mesh element;
- data values, which are associated with this mesh element; and
- area, which corresponds to the actual area of the mesh element (2D case), or to its volume (3D case).

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

Resolutions and Representations of Main and Subregions

We store the two numbers that define 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
The trees are designed in such a way that each leaf 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, a subscribe i is the area or volume associated with the i-th point, and f subscribe i is the function value associated with the i-th point.

Error
Concerning the averaging of function values, 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 neglect the resulting error due to the following argument: Since we are working with large data sets, we are concerned with large numbers of mesh cells. Having this in mind, and assuming that the area or volume of the whole data set is some constant, we can assume that the area or volume of a single cell is relatively small. It makes little difference whether one considers a cell that lies only partially inside a box as lying entirely inside or outside. Therefore, 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.

However, there is a second error measure, which we determine. Many points of the function values of the original may dramatically differ from the average. We compute two errors for the bounding box, the root mean square (RMS) error and the maximum error (MAX):
where F is the average of all function values associated with the box.

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

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

To visualize the error of an approximation, we use gray-scale shading. White corresponds to zero error, and black corresponds to the highest error (Figures 4 and 5).

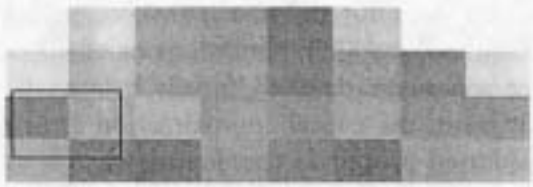

FIGURE 5.
RMS error for main and subregions.

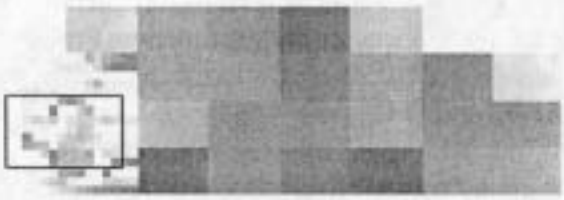

FIGURE 6.
Example of an irregular grid.

To reduce the error we decrease the number of points per box, making 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. Provided that a region is small enough so that it contains at most one point, ideally we have zero error.

Although the error calculation might look like a very time-consuming operation, it is necessary to do it. The use of the error function can speed up the whole algorithm. In a near constant-value region, the error will be smaller than some tolerance e. In this case usually only a small number of subdivisions is needed. In other words, we can stop subdividing such a region before we reach the maximally allowed depth of the tree.

Applications

To make use of our data structure, we initialize the set of points in our data structure from an input data set and build the main octree that corresponds to the main region. The depth of the main tree corresponds to the resolution of the main region. Typically, the depth of this main tree is very low since we do not want to represent the main region in great detail. Both initializing the set of points and computation of the main octree structure are straightforward (Figure 7). After constructing the main tree, we can display the main region at low resolution, and the user can select a subregion (ROI).

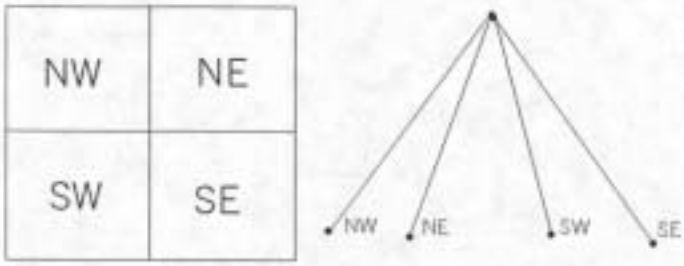

FIGURE 7.
Main region and corresponding quadtree (2D case).

We then construct a tree that corresponds to the selected subregion. To do this efficiently we take advantage of the fact that the points are already sorted in the main tree. Having this in mind and using the main tree as a search tree, we can easily find a node A whose bounding box contains our subregion. Thus, to build the sub-octree, we do not consider all the points, as we did for building the main tree, but only those points that are inside the bounding box of node A. This allows us to process a smaller number of points and speed up the algorithm (Figure 8).

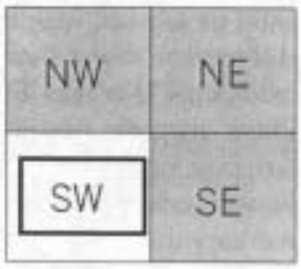

FIGURE 8.
Subregion inside main region not including center of main region.

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.

However, if the subregion is near the center, the node whose bounding box contains the whole subregion might be only the root. That makes it necessary to re-process all points again for building the sub-tree. There are several options to deal with this problem.

First of all, if we know beforehand that the ROI is located closed to the center of the scene, we can subdivide the space at an uneven ratio (for example 1/3 or 2/3) when we construct the main tree (Figure 9). Besides this, even if node A happens to be the root, we can traverse down from node A towards the leaves and purge points outside the subregion by ignoring all branches and leaves whose bounding boxes do not touch the ROI (Figure 10).

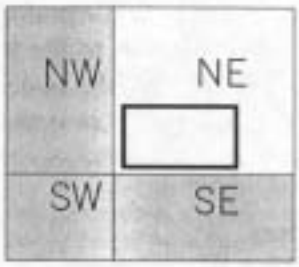

FIGURE 9.
Subregion inside main region including center of main region.

(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.)

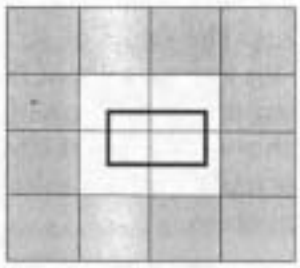

FIGURE 10.
Subregion inside the main region including center of main region.

(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.)

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.

Note that the root has a bounding box that includes all points of the region, 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 those 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

Let us consider the data set that is 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=(Error! Unknown switch argument.** di / N)1/2, 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 the RGB value of the i-th point varies from the average.

Tree Depth and Error

Increasing the depth of the tree decreases the error. Considering the data set shown in Figure 1, these are our results: A main tree depth of two leads to an average value of the RMS errors of 140, and the depth of four leads to an RMS error of 59. Thus, increasing the depth by factor of two, we decrease the RMS error more than twice.

Concerning Figure 3, a sub-tree of depth two corresponds to an RMS error of 132, and a depth of four corresponds to 66. 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 the rest. 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 can 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 tech-

nique 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. pinskiy@cs ucdavis.edu

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. jmeyer@cs ucdavis.edu

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. hamman@cs ucdavis.edu

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. joy@cs ucdavis.edu

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. brugger@llnl.gov

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. duchaineau@llnl.gov