Constructing isosurfaces in a localized fashion using an underlying octree data structure

Dmitriy V. Pinskiy[a], Eric S. Brugger[b], Sean D. Ahern[b], and Bernd Hamann[a]

[a]Department of Computer Science, University of California, 1 Shields Avenue, Davis, CA 95616
[b] Lawrence Livermore National Laboratory P.O. Box 808 - Mail Stop L-98, Livermore, CA 94550

ABSTRACT

We present an octree-based approach for isosurface extraction from large volumetric scalar-valued data. Given scattered points with associated function values, we impose an octree structure of relatively low resolution. Octree construction is controlled by original data resolution and cell-specific error values. For each cell in the octree, we compute an average function value and additional statistical data for the original points inside the cell. Once a specific isovalue is specified, we adjust the initial octree by expanding its leaves based on a comparison of the statistics with the isovalue. We tetrahedrize the centers of the octree's cells to determine tetrahedral meshes decomposing the entire spatial domain of the data, including a possibly specified region of interest (ROI). Extracted isosurfaces are crack-free inside an ROI, but cracks can appear at the boundary of an ROI. The initial isosurface is an approximation of the exact one, but its quality suffices for a viewer to identify an ROI where more accuracy is desirable. In the refinement process, we refine affected octree nodes and update the triangulation locally to produce better isosurface representations. This adaptive and user-driven refinement provides a means for interactive data exploration via real-time and local isosurface extraction.

Keywords: Octree, isosurface, region of interest, local refining, triangulation, volume visualization

1. INTRODUCTION

Isosurface extraction and rendering is a fundamental visualization method for scientific data exploration. However, in most scientific field, including, for example, medical imaging, vector field, and flow simulation, data sets have grown in size to several million mesh elements. Considering finite element/difference simulations, the extraction of an isosurface can become prohibitively expensive and intractable. Moreover, producing high-detailed isosurface renderings could be wasteful if a user is interested in only a small portion of an isosurface.

A solution to the above problem is to apply different levels of detail to isosurface extraction and rendering. First, we construct a rough approximation of an isosurface that serves as a preview for a user. The user can then identify an ROI where he/she desires locally more precise isosurface extraction.

2. RELATED WORK

Considering certain error estimates, we "fit" an octree representation to a given data set. A related approach is described by Laur et al.[1] 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 by Hamann et al.[2,3,4] deal with the problem of decimating triangular surface meshes and adaptive refinement of tetrahedral volume meshes, respectively. These decimation approaches are aimed at the concentration of points in regions of high curvatures (or high second derivatives), and they can be used to eliminate points in nearly linearly varying regions (decimation) or 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 in regions of large error or when a user requests local refinement.

Further author information: (Send correspondence to D.V.P.)
D.V.P.: E-mail: pinskiy@cs.ucdavis.edu
E.S.B.: E-mail: brugger1@llnl.gov
S.D.A.: E-mail: ahern@llnl.gov
B.H.: E-mail: hamann@cs.ucdavis.edu

We use the difference between the original data values and the piecewise constant-value octree approximations, considering each octree cell, as an error metric. In our application, the term data-dependent discretization refers to decompositions of space that approximate a continuous function by a piecewise constant function. Previous work on data-dependent discretization techniques is covered by Dyn et al.[5,6] and Schumaker.[7]

In principle, our technique relates to an idea of constructing a "data pyramid." A data-pyramid is a data hierarchy of cells with increasing precision.[8] The pyramid concept has also been extended to the adaptive construction of tetrahedral meshes for scattered scalar-valued data.[9,10] 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 by DeRose et al.[11] and Eck et al.,[12] for example. Our data-dependent, octree-based 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 by Preparata et al.[13]

3. MAIN APPROACH

In this section, we outline and discuss the major steps of the proposed method.

3.1. Octree Construction

We assume that the input to our method is a set of scattered data: points in space with an associated scalar function value. (For the purposes of our method, we ignore possibly known point connectivity information.) The first step of our algorithm is octree construction. First, we compute the bounding box of the set of all originally given points. This bounding box is then subdivided in an octree fashion until each leave node in the octree satisfies a specific termination criterion for refinement. We consider two types of termination criteria: (i) a node is no longer split when the number of original points lying in it is smaller than some threshold number or (ii) a node is no longer split when the error between the function values of the original points inside the node and an average function value associated with the node is smaller than some error threshold. One of the main purposes of the construction of the initial octree is reducing the storage requirements to represent the given scalar field at a coarse resolution. The octree, eventually, will also greatly reduce the time necessary to produce good isosurface approximations.

3.2. Tetrahedral Mesh Construction and Isosurface Approximation

How is the initial octree data structure used to define a first, crude approximation? Considering the function values of all the original points inside each leaf node, we compute an average function value for the leaf node. This average function value is then associated with either the center of the leaf cell or with the average coordinates of all the points inside the leaf. We will refer to the points with which we associate the average function value as "center leaf points." One has to consider the special case when a leaf does not contain any points: In this case, we propose to consider points from neighboring leaves to determine an appropriate function value for the "empty" leaf.

How do we utilize the octree data structure to extract an initial, crude isosurface triangulation? Since the faces of the leaves in an octree usually define partial-face but not full-face matching, it is not possible to extract "crack-free" isosurface triangulations directly. To overcome this problem, we need to construct a tetrahedral mesh that is implied by the octree.

We triangulating (-or "tetrahedrize") the center leaf points and determine the tetrahedral meshes that decompose the entire space of interest in a full-face matching fashion. It is thus possible to extract an initial crack-free isosurface approximation where the isosurface is represented by a set of three- and four- sided polygons. (Each polygon is a tile that approximates the intersection between a tetrahedral mesh element and the isosurface.) This initial isosurface will, generally, be merely a rough approximation of the exact one, and its main purpose is to guide a viewer in identifying regions in space where a more accurate isosurface representation is desirable. This kind of adaptive and user-driven refinement is a part of the interactive data exploration process supporting real-time isosurface refinement operations.

3.3. Defining and Moving an ROI

After a first isosurface approximation is generated, a user can specify regions of particular interest by, for example, creating local bounding spheres or bounding boxes. Once specified, these bounding elements indicate to the system that a higher level of accuracy of isosurface representation is demanded in the affected regions in space. At this point, the space coordinates of the ROI will be translated into the octree coordinates (the octree location code) to identify the octree leaf nodes covering the ROI. These leaf nodes would be subdivided to achieve a better approximation.

In practice, to promote further subdivision, we would specify smaller values N and E in these nodes (where N is a required minimum number of points inside a cell, and E is the error between the function values of the original points and an average function value associated with the node). Thus, the initially created octree will be refined only in regions of greater interest. Once the octree has been refined locally, we update our set of center leaf points; and then, using an incremental version of Delauny triangulation,[14] we can relatively quickly update the triangulation. As a result, the tetrahedral mesh is refined and updated locally. The local refinement process will impact the geometry of the extracted isosurface triangulation only locally. Therefore, the changes of the isosurface will not impact the rendering performance significantly.

The adaptive isosurface extraction method supports both local refinement and coarsening operations. Coarsening, in contrast to refinement, allows a user to "merge" leaf nodes into a larger one, thus reducing storage requirements. Coarsening will therefore be a means for accelerating the rendering process. By combining local coarsening and refinement operations, a user can navigate the entire space of interest in real time while keeping memory requirements for the octree data structure low.

4. ADDITIONAL IMPROVEMENTS

The additional improvements discussed in this section are mainly aimed to achieve a decent "approximation of C1 continuity" and to avoid construction of large octrees.

4.1. Striving for C1 Continuity and Angle Tolerance

When we extract an isosurface, we want to "preserve C1 continuity," or at least, "approximate" it. On the other hand, we do not want to introduce the appearance of continuity in regions where it is not present in an isosurface.

A naive approach to accomplish that would be simply to refine the whole octree by making the tree as deep as possible. That would result in the finest triangulation, which leads us to a high-precision isosurface approximation. However, this approach would result in an enormous increase in rendering complexity and memory requirements for our data structure. (Certainly, this solution would also overkill the whole purpose of the multiresolutional approach for visualization of large-scale data sets.)

A better approach would be to refine only regions where we absolutely have to–i. e., where the difference between the true isosurface and its approximated representation is higher than some tolerance. For example, consider Fig. 1 that represents isoline construction in the 2D, bivariate setting, one note the following:

- Refinement of mesh 2 would be unnecessary and wasteful.
- The approximation needs improvement in meshes 4 and 5 by applying further refinement.

The question that we address is: How can one identify the regions that need further refinement? Let us observe some mesh M that the approximated isoline (2D case) or isosurface (3D case) passes through. Since the isoline or isosurface is represented inside the mesh by line segments or plane segments, respectively, C1 continuity is always preserved inside the mesh. However, our isoline/isosurface approximation can be C1-discontinuous at the boundaries of the mesh. (For example, consider mesh 4 in Fig. 1.)

Let us consider the bivariate case first. The C1 discontinuity can be identified by noticing that the isoline approximation changes its direction by a large angle. Therefore, by calculating angles between individual approximating segments of neighboring meshes, we can locate regions that need refinement. Considering Fig. 1, we can see that mesh 4 needs refinement since the approximated isoline changes its direction in the neighboring meshes 3 and 5, while mesh 2 does not need refinement because the approach of the isoline has consistent behavior in the neighborhood of the mesh. Fig. 2 shows the isoline approximation after refinement. We can summarize the approach in the following high-level pseudocode for the bivariate case:

```
For each mesh M through which the approximated isoline I passes
    For each neighbor Ngb of M where Ngb is such that I passes through Ngb
        Let S1 and S2 be line segments that approximate I in the M and Ngb;
        If angle between S1 and S2 is greater than the angle tolerance T
            Refine M;
```

The angle tolerance is a maximum allowed value of the angle between isoline approximations (segments) of two neighboring meshes that form one continuos piece of the isoline. Thus, smaller angle tolerances will enforce the creation of better approximations in terms of C1 continuity. An angle tolerance can be different for different regions of our data set, depending on importance of the region for the user. For example, a user might specify a low angle tolerance for a specific ROI, while, to keep computation time low, he or she might not use C1 improvement at all for certain regions that are preserved primarily for context information.

A similar logic can be used in the trivariate case. However, when we must determine whether to refine further or not, we should measure angles formed by polygon normals of the approximating tiles inside neighboring tetrahedral mesh elements.

If, after a series of refinements in some region, we have approximately the same magnitude of angle "variation" between line segments or normals of tiles, then that would suggest that the true isosurface is discontinuous, and we should not further refine the region to achieve move smoothness. Thus, this approach prevents us from introducing C1 continuity as well as from doing wasteful computations.

4.2. Global Refinement Based on Statistics

When we construct our initial octree, we face an obvious dilemma about the height of the octree. An octree that is only a few levels deep would not be sufficient for a user to identify an ROI; on the other hand, expanding deep nodes of the tree might be useless and take up significant memory resources.

We approach this problem by building first an octree with relatively short height. This can be done as a part of the preprocessing stage and without knowing the isovalue. When knowing the isovalue or, at least, having an idea about possibly good isovalue, we adjust our octree to the isovalue by expanding appropriate nodes.

Our approach is based on the observation that it is wasteful to expand octree nodes that correspond to regions lying far away from our isosurface or regions that contain only "noise" rather than significant parts of the isosurface. Therefore, by recursively constructing an octree, we should be able to decide whether we need to subdivide a currently considered octree node further based on the prediction how the true isosurface would look like in the region corresponding to this node. To perform this prediction without actual isosurface construction, we take advantage of statistics calculated for each node. These statistics include:

- minimum and maximum function values of the points inside each node and
- variance / standard deviation of function values of the points inside each node.

The calculation of these parameters should not slow down the octree construction since we can use these data also for the second termination criterion (a node is no longer split when the difference between the function values of the points inside the node and an average value of the points is smaller than some user-defined tolerance). Our approach can be expressed in the following high-level pseudocode:

```
If isovalue $\in$ [node.min; node.max]
    If isovalue $\in$ [node.average $-$ n $\times$ node.variance; average $+$ n $\times$ node.variance]    /* n = some integer */
        Subdivide the node further;
```

A simple check whether an isovalue is in the minimum-maximum range is not sufficient because we might end up with some noise that is not worth of expensive subdivision of the node. Therefore, we perform a different, second operation to ensure that there is a number of points with associated function values that are greater and smaller than the isovalue. This is done by using the integer parameter n that should be proportional to the number of points in the cell. Assuming a uniform distribution of points in space, we can approximate n by making it proportional to the level of the node in the octree.

5. IMPLEMENTATION AND RESULTS

In this section, we propose a special approach to select center leaf points for triangulation purposes in order to gain additional efficiency in the implementation. We also discuss some data structure issues.

5.1. Analytical Selection of Points

Our results have shown that one of the most time-consuming parts of the algorithm is triangulation (bivariate case) or tetrahedrazation (trivariate case). This becomes a very important bottleneck, especially when performing a preprocessing step for building an expanded octree. For example, one might want to first run a program overnight to construct the expanded octree that is equally oriented for any isovalue, and then, knowing the octree, interactively explore the data set in real time by setting different isovalues and specifying ROIs on an isosurface. We end up with a large number of leaf nodes that would translate into a big number of points to triangulate.

The key observation to solve this bottleneck that not all meshes are equally important for isosurface approximation. Meshes consisting of points with function values that are all greater than the isovalue (or all smaller than the isovalue) would not be used directly in isosurface construction because the approximated isosurface would not cross them. Therefore, we should prevent our algorithm from spending time for constructing such "useless" meshes.

To accomplish that goal, we need to note that center leaf points, which have function values greater than the isovalue and are surrounded by only center leaf points with values also greater than the isovalue, would result in "useless" meshes. We can solve the problem by detecting such points and simply not inserting them into our triangulation. (Exactly the same logic applies if the value of a center leaf node is less than the isovalue.)

Fig. 3 illustrates the basic idea of the approach. It shows a quadtree decomposition of the two-dimensional space. Black points correspond to points with function values greater than the isovalue, and white points represent points with function values less than the isovalue. Any center leaf point of colored nodes is surrounded by only black points and leads to the creation of meshes that the isoline would not intersect. Therefore, we can improve running time behavior by not including the points from the colored nodes in the triangulation. Our approach can be summarized like this:

```
For each leaf node N
    For each N's neighbor Ngb
        If (Ngb is a leaf AND
          (Ngb.value $\leq$ Isovalue $\leq$ N.value OR N.value $\leq$ Isovalue $\leq$ Ngb.value))
                Insert N into triangulation;
                Exit from both loops;
        If (Ngb is a branch AND
          (Ngb has a leaf L such that L is a neighboring cell for N AND
          (L.value $\leq$ Isovalue $\leq$ N.value OR N.value $\leq$ Isovalue $\leq$ L.value)))
              Insert N into triangulation;
              Exit from both loops;
```

Checking for existence of the leaf L may take a long time, especially when Ngb is close to the root. Therefore, to improve time performance, we can approximate this checking by simply considering variance, minimum and maximum values associated with Ngb and performing a test similar to the one described in the section about global refinement based on statistics. Another solution, which would be much safer and simpler, is to always insert N when at least one of its neighbors is a branch.

In practice, this careful selection of points for triangulation purposes can increase performance substantially, especially when we aim to construct a high-quality isosurface approximation. As an example, consider Fig. 4 showing the true isosurface—a sphere. Points that are inside the sphere have function values equal to one, and the function values of points outside the sphere are zero. Generating a crude isosurface approximation on a desktop PC (475MHz processor clock) using 125 central leaf points requires only a few seconds, regardless of whether we use the optimization (see Fig. 5). On the other hand, when we attempt to produce a better approximation, using about 36,000 leaf points, the triangulation step takes more than thirty minutes while, when choosing only necessary leaf points to triangulate, the triangulation time including time needed for selection is less than a minute; and we can produce exactly the same isosurface (see Fig. 6).

5.2. Data Structure

We implemented our volumetric data set representation as two octrees—global and local. The global octree represents the whole data set. We used the centers of this octree's leaves in the triangulation to produce a crude approximation of the whole isosurface. The local octree covers only an ROI and is used to extract a more precise isosurface locally—only inside the ROI.

Let us compare these two trees. They have approximately the same depth. However, since the local octree covers less space, each of its nodes approximates less points than the global one does. As a result, the local octree yields a better isosurface triangulation. Fig. 7 shows an approximation of the whole spherical isosurface, and the ROI is shown in the upper-right corner. The depths of the global and local octrees are six and eight, respectively. While these depth-values are pretty similar, we notice a much better approximation of the spherical isosurface in the ROI than in the main region.

To construct the global octree, we need to deal with the whole data set, and that might be very time-consuming. Therefore, we might preprocess the data set and build a global octree over night. In contrast, building a local octree should be done in real time since specifying and navigating an ROI is a part of user-driven, interactive data exploration process. We can accomplish real time by taking 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 the ROI. 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 constructing the local octree. The most essential pieces of our octree are its nodes. Each node consists of the following fields:

- a pointer to the parent node;
- flags to indicate whether a node is a non-empty 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. Leaves have the smallest bounding boxes in the tree.

When a node is a branch, it also has fields such as a pointer to its children. 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. (As a relationship between a node and a branch and a relationship between a node and a non-empty leaf might suggest, we should derive both a branch and a non-empty leaf data structure from a node data structure.)

6. CONCLUSION

Employing an error-controlled octree data structure, our algorithm allows a user to explore large-scale scientific data sets by building, first, a crude approximation of the isosurface, and then locally refining regions that are important for a user. This algorithm has great potential, including support of multiple and nested ROIs. We believe that adaptive approaches, like the one we presented here, will play an increasingly important role in future large-scale scientific data exploration.

ACKNOWLEDGMENTS

We thank especially Hank Childs from B Division of Lawrence Livermore National Laboratory, which mainly supported this work. This work was also supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), 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; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

REFERENCES

1. D. Laur and P. Hanrahan, “Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering,” in proc. *SIGGRAPH (Computer Graphics)*, T. W. Sederberg, ed., **25**, pp. 285-288, Association for Computing Machinery Inc, New York, 1991.
2. B. Hamann, “A data reduction scheme for triangulated surfaces,” Computer Aided Geometric Design, **11(2)**, pp. 197-214, 1994.
3. B. Hamann and J. L. Chen, “Data point selection for piecewise linear curve approximation,” Computer Aided Geometric Design, **11(3)**, pp. 289-301, 1994.
4. B. Hamann and J. L. Chen, “Data point selection for piecewise trilinear approximation,” Computer Aided Geometric Design, **11(5)**, pp. 477-489, 1994.
5. N. Dyn, D. Levin, and S. Rippa, “Data dependant triangulations for piecewise linear interpolation,” IMA Journal of Numerical Analysis, **10**, pp. 137-154, 1988.
6. N. Dyn, D. Levin, and S. Rippa, “Algorithm for the construction of data dependent triangulations,” in it Algorithms for Approximation II, Mason, J.-C., ed., pp. 185-192, Chapman and Hall, New York, 1990.
7. L. L. Schumaker, “Computing optimal triangulations using simulated annealing”, Computer Aided Geometric Design, **10(3-4)** pp. 329-345, 1995.
8. L. De Floriani, “A Pyramidal Data Structure for Triangle-Based Surface Description,” IEEE Computer Graphics and Applications,**9(2)**, pp. 67-78, 1989.
9. M. Bertolotto, L. De Floriani, and P. Marzano, “Pyramidal simplicial complexes,” in proc. *Third Symposium on Solid Modeling and Application*, C. Hoffmann and J. Rossignac, eds., pp. 153-162, Association for Computing Machinery Inc, New York, 1995.
10. P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno, “Multiresoultion modeling and visualization of volume data based on simplicial complexes,” in proc. *ACM Symposium on Volume Visualization*, A. E. Kaufman and W. Krueger, eds., pp. 19-26, Association for Computing Machinery Inc, New York, 1994.
11. A. D. DeRose, M. Lounsbery, and J. Warren, “Multiresolution analysis for surfaces of arbitrary topological shape,” Tech. Rep., Department of Computer Science and Engineering, University of Washington, Seattle, 1993.
12. M. Eck, A. D. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, “Multiresolution analysis of arbitrary meshes,” in proc. *SIGGRAPH (Computer Graphics)*, R. Cook, ed., **29**, pp. 173-182, Association for Computing Machinery Inc, New York, 1995.
13. F. P. Preparata and M. I. Shamos,*Computational Geometry*, Springer-Verlag, New York, 1991.
14. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf,*Computational Geometry”*, Springer-Verlag, New York, 1997.

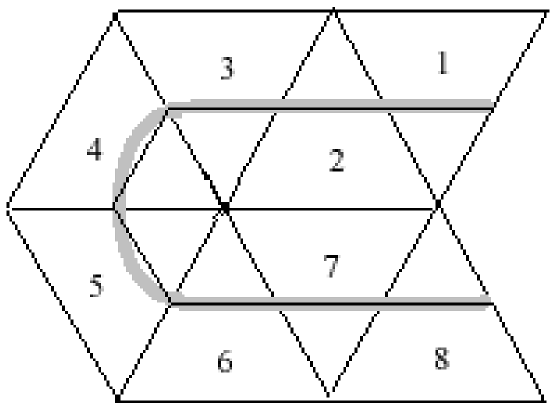

Figure 1. Construction of crude approximation of isoline: gray line represents true isoline, and black line corresponds to isoline approximation implied by triangular meshes.

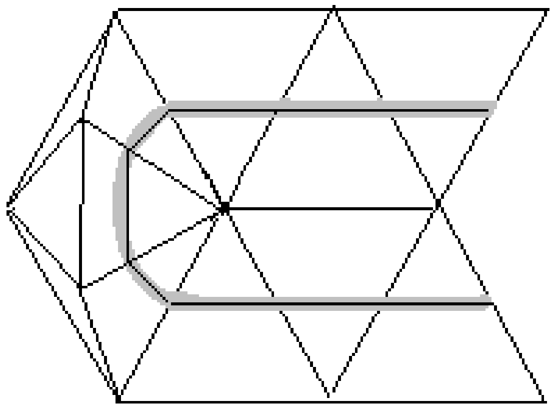

Figure 2. Refining approximation of isoline by further subdivision of triangular meshes: gray line represents true isoline, and black line corresponds to isoline approximation implied by triangular meshes.

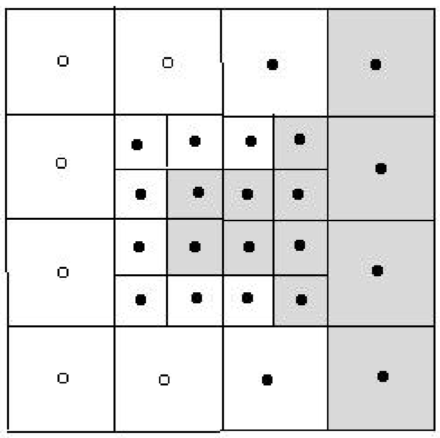

Figure 3. Quadtree decomposition of space: black dots represent center leaf points with function values greater than the isovalue, while white dots correspond to center leaf points with values less than the isovalue. Points in the colored area should not be used for triangulation.

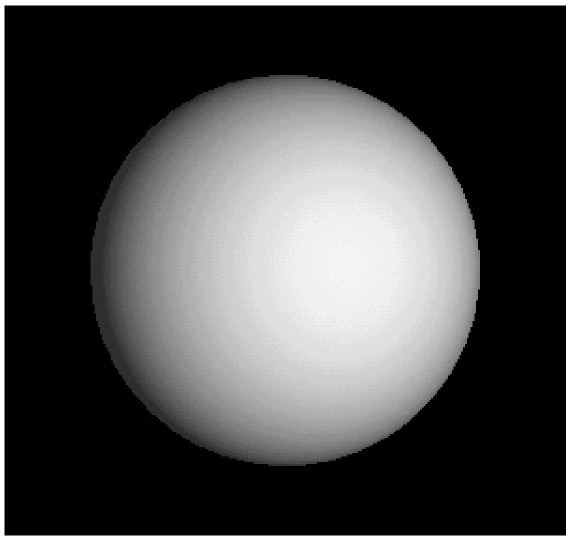

Figure 4. True isosurface, a sphere.

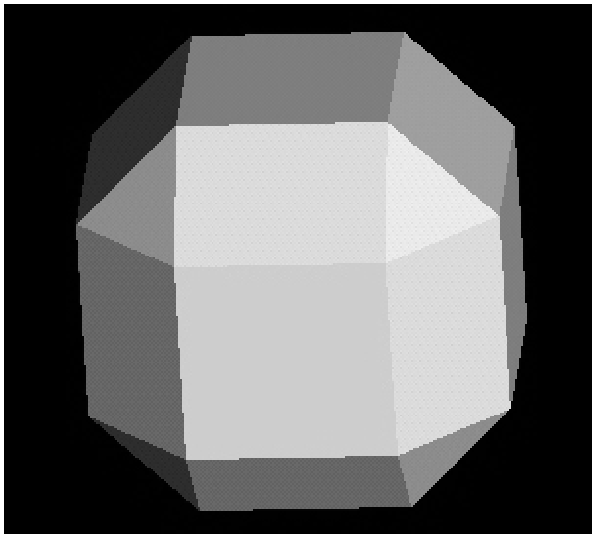

Figure 5. Approximation of spherical isosurface based on 125 center leaf points.

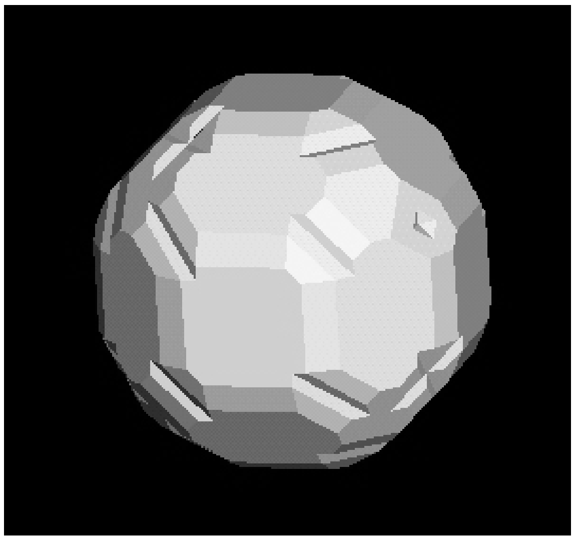

Figure 6. Approximation of spherical isosurface based on approximately 36,000 center leaf points.

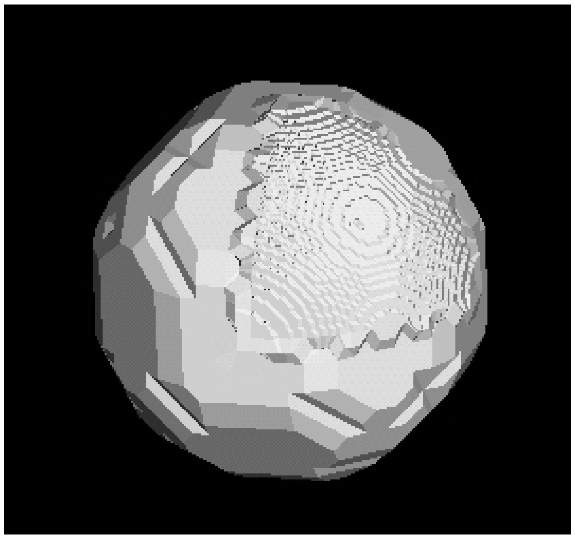

Figure 7. Approximation of ROI based on approximately 72,000 center leaf points along with approximation of spherical isosurface based on approximately 36,000 center leaf points.