Real-time View-dependent Extraction of Isosurfaces from Adaptively Refined Octrees and Tetrahedral Meshes

David C. Fang, Jevan T. Gray, Bernd Hamann, and Kenneth I. Joy

Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science
University of California, Davis
Davis, CA 95616
{fangd, grayj, hamann, joy}@cs.ucdavis.edu

ABSTRACT

We describe an adaptive isosurface visualization scheme designed for perspective rendering, taking into account the center of interest and other viewer-specified parameters. To hierarchically decompose a given data domain into a multiresolution data structure, we implement and compare two spatial subdivision data structures: an octree and a recursively defined tetrahedral mesh based on longest-edge bisection. We have implemented our view-dependent isosurface visualization approach using both data structures -- comparing storage requirements, computational efficiency, and visual quality.

Keywords: view-dependent, isosurface, multiresolution, tetrahedral mesh, octree, hierarchical

1. INTRODUCTION

Interactive visual exploration of volumetric scalar field data is often done through isosurface extraction and rendering. As discrete scalar field data sets have increased substantially in size over the past decade, it is desirable to have an interactive visualization framework available that supports the procedural, real-time extraction of isosurfaces, considering specific viewing parameters such as line of sight (LoS) and distance from the eye. We describe a framework that allows a viewer to interact with progressively refined isosurface triangulations, depending on the center of interest and other view-specific parameters.

To facilitate real-time viewing, we present two types of adaptive mesh structures and an "oracle" for guiding mesh refinement. Adaptive meshes allow for regions of greater importance to be refined to finer levels while leaving other areas coarse. For our purpose, regions closer to the eye are considered more important. Furthermore, a progressive refinement strategy can be used to ensure rapid frame updates, even when eye position, and thus region of interest, is changed.

2. OBJECTIVE AND MOTIVATION

Performing a full-resolution isosurface extraction algorithm for very large datasets has two main drawbacks: It is slow, and it produces a large number of surface elements that are often nearly co-planar. The number of nearly co-planar elements can be reduced by applying an additional simplification step. Even after simplification, isosurfaces are often so complex and large that their storage requirements are greater than those of the original datasets, making the idea of storing extracted surfaces less feasible. Furthermore, there are often several different isosurfaces/isovalues of interest in a dataset, which contributes additionally to the "data explosion." It is also extremely time-consuming to render the large sets of isosurface elements.

These observations motivated us to develop a real-time isosurface viewing method based on the principle of perspective viewing. Only the original dataset is stored, and a desired isosurface can be extracted and rendered in real time. To support real-time system behavior, we cannot always use the dataset at full resolution when calculating an isosurface.

Polygons further away from the eye are projected to relatively fewer pixels in the frame buffer than closer polygons of the same size. We take advantage of this property by leaving distant isosurface polygons at relatively lower resolutions and adaptively refining the regions close to the center line of attention. This approach allows for a reduction in the number of polygons rendered while being able to control the visual quality. The size of polygons produced by isosurface extraction techniques can be limited by the size of a mesh cell. The size of a cell, when projected to the window, is a user-specified parameter, called maximum-pixels-per-cell (MPPC), which can be used to adjust visual quality and computation time. The same principle can be applied to polygons further away from the line of sight. These paradigms allow us to reduce the number of polygons rendered while maintaining the desired level of visual quality. The technique can be used to guarantee a desired frame rate by varying cell size and using progressive mesh refinement.

To hierarchically decompose a given volumetric domain into a multiresolution data structure, we have implemented two spatial subdivision data structures: (i) an octree and (ii) a recursively defined tetrahedral mesh based on longest-edge bisection. The root of the octree is a single cube, defining the coarsest, base level. The octree is refined by splitting cubes into eight child cubes of equal size. In contrast, the base level of the tetrahedral mesh consists of six tetrahedra sharing the edge that is a diagonal of the bounding box (typically a cube) connecting two opposite vertices. Refining the tetrahedral mesh is done by bisecting each tetrahedron along its longest edge. We have implemented the view-dependent real-time isosurface visualization approach using both subdivision schemes to compare the two data structures. Storage requirements, computational efficiency, and visual output quality are discussed.

3. PREVIOUS WORK

Isosurface extraction is a common task, and many methods have been developed for/based on it. The original marching cubes algorithm[1] did not cover certain ambiguous cases. Solutions to the ambiguous configurations were developed, for example, by Nielson and Hamann[2]. Later, a tetrahedral-based marching algorithm[3], based on splitting each cube into five tetrahedra, was developed. It is simpler than marching cubes and does not lead to ambiguous configurations. Unfortunately, it introduces ambiguity due to the symmetry of the subdivision possibilities for a cube. Chan and Purisma[4] suggested a subdivision strategy that splits a cube into six tetrahedra, which removes the ambiguity. Optimizations of marching cubes focusing on reducing search time for finding cells containing the isosurface include the work by Cignoni et al[5]. Cignoni et al. demonstrated how to use the concept of interval trees to quickly identify cells containing the isosurface. Shekhar et al.[6] used surface tracking to find the next cells to traverse using an initial seed cell containing the connected component of the isosurface desired. Bajaj et al.[7] combined the idea of interval trees and seed growing by building the interval tree only for the seed set, which further reduces search time. Another focus of isosurface research has been the reduction of triangle count while preserving visual quality.

Hierarchical data structures allow for an adaptive level of detail, using full detail only in regions of the isosurface that need it, thus reducing triangle count. Shekhar et al.'s method starts with a fully refined octree and performs a bottom-up merging step of cells to reduce triangle count in areas where merging keeps the error below some tolerance. Starting at the finest level is a serious drawback when working with large datasets. Instead, Shu et al.[8] used an octree structure, starting with a uniform size, splitting cells until a curvature criterion is met for the surface passing through the cell. Zhou et al.[9] presented a tetrahedra-based hierarchical mesh that is the basis for our tetrahedral data structure. Their method first refines to the finest resolution level and then proceeds with a fine-to-coarse merging strategy to construct adaptive meshes. Exploiting the inherent parallelism of ray tracing, Parker et al.[10] achieved interactive ray tracing performance for isosurfacing on shared-memory supercomputer systems by massive parallelization. Livnat et al.[11] presented a view-dependant isosurface extraction method based on considering only the visible portion of an isosurface. A hierarchical visibility test is used in their method to determine regions of the volume that are occluded. The volume is decomposed using an octree and traversed in front-to-back order. The visibility test is performed using hierarchical tiles based on coverage masks, which makes it possible to extract only the visible portion of an isosurface. This approach often results in a greatly reduced triangle count, but also produces results that are incomplete when viewed from a different position.

4. ALGORITHM

Our algorithm determines an upper bound for the number of pixels that an object maps to. If the number of pixels is above the MPPC, the object is refined until it meets the user-specified pixel threshold. Our algorithm operates on a per-cell basis. For an octree, a cell is a cube, while a cell is a tetrahedron in a tetrahedral mesh. Cell refinement is determined by viewing, application, and user-specified parameters. Viewing parameters include the location of a cell relative to the eye and the field of view. The application parameters are the dimensions of the rendering area of the application (viewport). The MPPC and functions that limit the area of interest are parameters set by the user.

The upper bound for the number of pixels that a cell can map to needs to be calculated using the given parameters. For each vertex of a cell (four in the case of a tetrahedron and eight in the case of a cube), we project the vertex to the window. Each window coordinate is used to obtain a two-dimensional, axis-aligned bounding box for the projected vertices, see Figure 1. The number of pixels contained in the bounding box is an upper bound for the number of pixels that the cell can project to on the window. Projection is done using standard graphics matrix transformations. To simplify the implementation we use the OpenGL function gluProject() to transform the world coordinates into window pixel coordinates. The function gluProject() allows for the viewing parameters and object transformations to be determined in one function call. If the number of pixels contained in the bounding box is greater than the MPPC, then the cell is refined until the threshold is met.

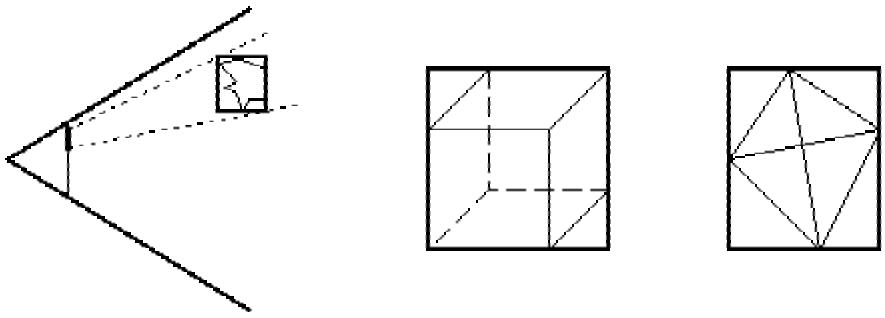


Figure 1. Cell projection onto viewport and bounding boxes.

Additional techniques are used to limit the region of interest and further reduce the triangle count. Cells relatively further away from the line of sight are often of less interest to a user and can be rendered at lower resolution. Cells that are completely outside of the viewing frustum can be left at the lowest resolution, see Figure 9. The angle between the line from the eye to the cell and the LoS can be calculated using trigonometric formulas. Instead of calculating the angle explicitly, we use the distance of the center of the bounding box to the center of the viewport. This distance is related to the angle of the LoS. Cells deeper in the volume are often of less interest and can be left at a lower resolution. Cell depth can be calculated using the depth buffer information that gluProject() provides.

The algorithm implements lower-resolution regions by artificially raising the user-specified pixel count as cells "drift away" from the center of attention. A coarsening function is used to describe the rate at which the pixel count is artificially raised, thus reducing cell resolution. As part of the specified parameters, either a linear or an exponential function may be used. An exponential function allows cells close to the center of attention to remain at the desired resolution while cells further away increase exponentially in size.

The algorithm presented does not actually calculate the number of screen pixels to be rendered per cell. To perform this computation would require a data-dependent scheme. We present an algorithm that guarantees an upper bound for the number of pixels that a cell can project to. This approach may not produce the optimal subdivision configuration for isosurfaces with large flat regions close to the center of attention. However, the frequency of such data could be taken into consideration when our algorithm is used.

5. IMPLEMENTATION

Our scheme can be used with data structures that support hierarchical refinement. In the case of a rectilinear grid, a hierarchical data structure can be overlaid on top of the rectilinear grid. We demonstrate our adaptive approach for two types of volumetric subdivision, one based on an octree and the other based on a tetrahedral subdivision scheme. The input to our method is a discretized volumetric scalar field data set, defined on an equidistant/uniform rectilinear grid with power-of-two resolution in all three dimensions. One dependent scalar value is associated with each vertex. The power-of-two property is required by the octree data structure and simplifies the implementation of the tetrahedral mesh scheme. Datasets that are not power-of-two are either interpolated to a power-of-two grid or embedded into the next-possible power-of two-grid.

Our method starts with a low-resolution data representation, using either the octree or tetrahedral mesh data structure. Cells are split when the underlying viewer-specified refinement condition determines that a split is needed. In consecutive frames, the eye position often changes only little, if at all. We can take advantage of this frame-to-frame coherence; for example, it is possible to use the same mesh from the previous frame as a starting point of the refinement for the next frame.

5.1 Octree Refinement

Each octree refinement step doubles the number of cells in every dimension. This leads to an overall factor-of-eight increase of cells for every refinement step. For each refinement step of a cell, 19 new vertices are introduced: one center vertex, six vertices for each face, and 12 vertices for each edge**,** see Figure 2**.** These refinements can generate regions where different-resolution cells share a common edge or face. In these regions, there exist "hanging nodes," where a vertex from a higher-resolution cell is not shared by a neighboring lower-resolution cell. These hanging nodes can cause "cracks," or discontinuities, in the rendered isosurface, see Figure 3. An additional step is required to fix cracks[6,8]. However, if the MPPC is small enough, the visual distortions of these cracks are barely noticeable.

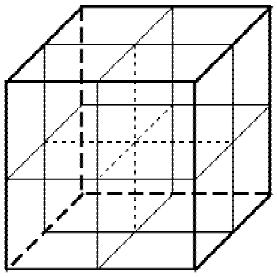


Figure 2. Octree subdivision

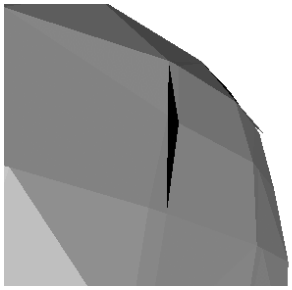


Figure 3. Crack in octree-based isosurface.

5.2 General/Uniform Tetrahedral Mesh Refinement

The tetrahedral mesh refinement scheme inserts the same points as the octree method, but fewer points per refinement step. In the first refinement step, "volume points" are inserted, in the next step "face points" are inserted, and finally "edge points" are inserted[12]. Thus, three steps of tetrahedral refinement are in a sense equivalent to one octree refinement step with respect to the number of points inserted. The tetrahedral mesh refinement step bisects the longest edge of every tetrahedron, splitting each tetrahedron into two. The set of tetrahedra sharing the split edge is often called "diamond." The point inserted by edge bisection is called the "dividing point," and it is the centroid of the diamond. Each step only doubles the number of vertices. There are three phases, or types of tetrahedra, in the mesh, phases 0, 1, and 2. Initially, the mesh consists of phase-0 tetrahedra. After three refinement steps we return to a phase-0 mesh, a refined version of the initial mesh. In a phase-0 mesh, six tetrahedra share their longest edge and make up a phase-0 diamond. The dividing point inserted when refining a phase-0 diamond is a volume point. The connectivity is adjusted after point insertion to form phase-1 tetrahedra, four of which share their longest edge and make up a phase-1 diamond. Bisecting again inserts the face points, which produces phase-2 diamonds from eight phase-2 tetrahedra. Finally, splitting the phase-2 diamonds inserts edge points and leads to a mesh of phase-0 tetrahedra and diamonds.

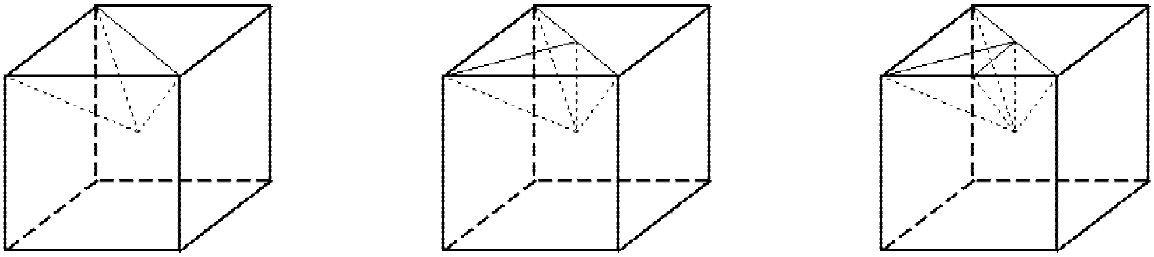


Figure 4. Three phases of tetrahedral refinement.

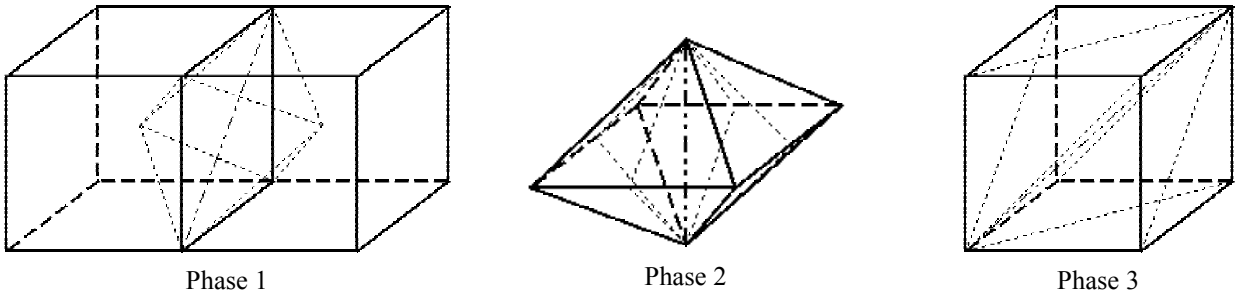


Figure 5. Diamonds associated with each refinement level.

5.3 Adaptive Tetrahedral Mesh Refinement

When splitting a tetrahedron, all tetrahedra in a diamond being considered must be split. When dealing with an adaptively refined grid, it is possible that some of the tetrahedra constituting the diamond do not belong to the mesh. A tetrahedron from a different phase can share the split edge, see Figure 6. This situation can occur when some diamonds were refined while others were not. To handle such a case we must first split the level-r diamond, leading to a "forced split," so that all the tetrahedra in the finer diamond are present. Sometimes the coarse diamond will be missing necessary tetrahedra, and an even coarser diamond must be split. This strategy is implemented via a recursive function. The forced splits ensure consistency of the interfaces of the tetrahedra, preventing "hanging nodes" and cracks during isosurface extraction. The gradual transition between refinement levels due to forced splits can be seen in Figure 9b. Similarly, diamonds are merged only when all of the tetrahedra of the diamond are present in the current mesh and satisfy the merging conditions.

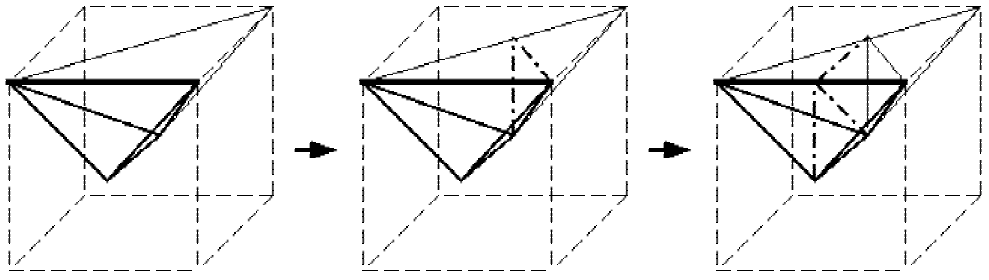
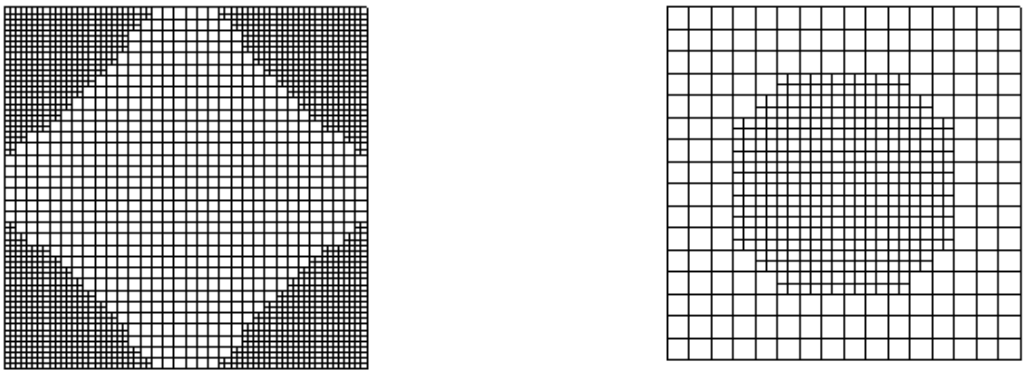


Figure 6. Forced split

6. RESULT



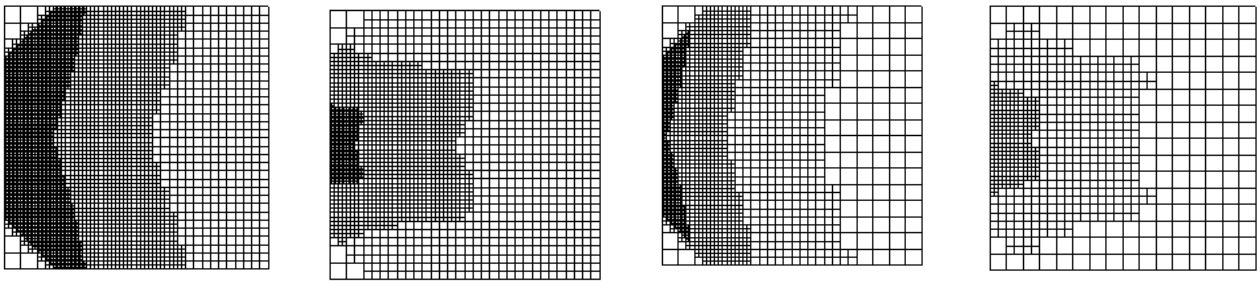
a) No coarsening

b) LoS coarsening

Figure 7. Meshes obtained with and without line of sight (LoS) coarsening (front view).

Figure 7a shows the resulting octree mesh with only the front surface cells displayed using no coarsening function. Note that the sizes of the cells further away from the center of the grid are more highly refined than the center cells. Though these cells are further away from the eye, they have the potential to project to more pixels. The reason is this one: The largest projection generated by a marching cubes algorithm in an axis-aligned cube is a plane that bisects the angle between any two axes and is parallel to the third axis. A viewing angle perpendicular to this plane would project to the greatest number of pixels. Cells that are not aligned with the LoS can also project a larger bounding box than axis-aligned cells having the same distance when viewed in the direction of the axis. These larger bounding boxes contain more empty space where extracted isosurfaces cannot be rendered to, see second and third pictures of Figure 1. A convex hull can be used instead of a bounding box to eliminate the empty regions, but such an approach would be computationally expensive for very large meshes. For reasons of efficiency and to enforce an upper bound on the number of pixels, these overestimations are acceptable. The effects of mis-aligned cells also apply to the tetrahedral mesh.

A coarsening function depending on the LoS can be used to limit the region of interest and allow cells further away from the line of sight to remain more coarsely represented, see Figure 7b. A coarsening function based on the depth of a cell in the viewing frustum can be used to further limit the region of interest. The distance from the near clipping plane to the farthest portion of the data volume is measured and used as either a quadratic or a linear scaling function for coarsening. Results obtained with both coarsening functions are shown in Figure 8.



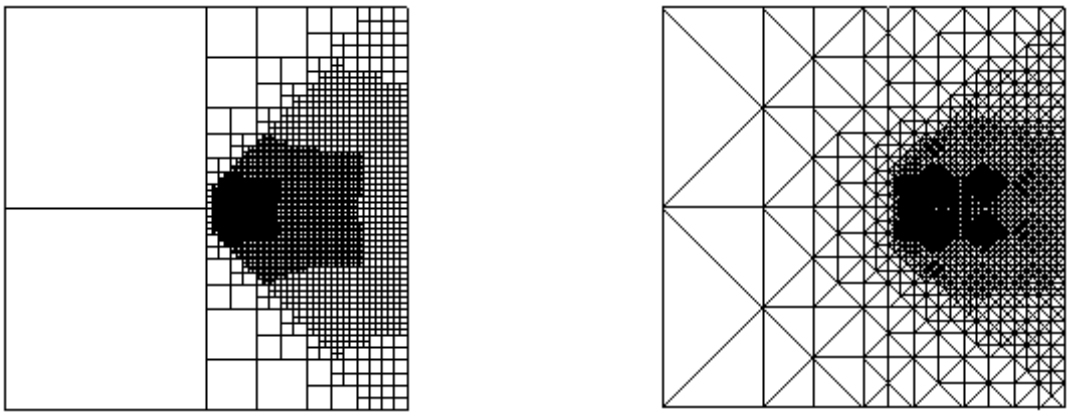
a) No coarsening

b) LoS coarsening

c) Depth coarsening

d) LoS + depth coarsening

Figure 8. Effects of different coarsening functions on octree mesh (side view).

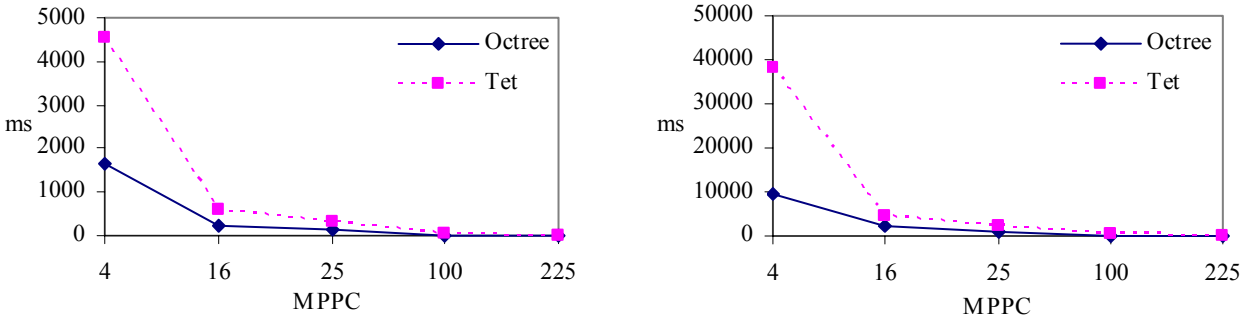


a) Octree mesh

b) Tetrahedral mesh

Figure 9. Octree and tetrahedral mesh refinement for a viewpoint inside the data volume looking right. Note how regions outside the view frustum are made coarse.

The speed of our algorithm depends partially on the MPPC. Selecting a larger value allows for lower-resolution cells to be used throughout the data set, resulting in substantially accelerated times. The listed refinement time is the time to refine a mesh starting from the coarsest level. Typically, the refinement process starts with the previous frame's refined mesh, which often only requires relatively minor modifications. Thus, the refinement time shown is much larger than the average, frame-to-frame refinement time. The extract + rendering time is the time to extract the isosurface and render it to the display using flat shading. Figure 10 shows the resulting times obtained from an Intel Pentium 4 2.0 GHz running with 2 gigabytes of RAM and a Nivida GeForce Ti 4600 running on the 256^3 skull dataset. The addition of coarsening functions leads to a further reduction in overall cell count by leaving regions of less importance at lower resolutions, see Figure 11. This reduction makes possible an additional frame rate increase. Once a user has selected a suitable viewing position and isosurface value, the MPPC can be adjusted to obtain an appropriate cell size and level of detail. A progressive refinement scheme could amortize the refinement times over multiple frames.



a) Extract + Rendering time

b) Refinement time

Figure 10. Effect of MPPC on computation time for a 256^3 skull data set.

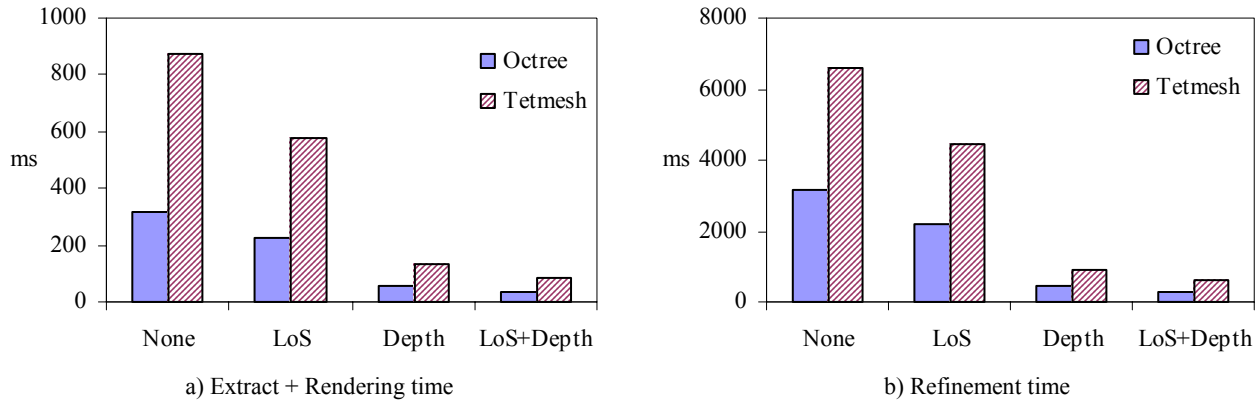


Figure 11. Effects of coarsening functions on performance for a 256^3 skull data set, MPPC = 16.

The viewport size greatly affects the speed and visual quality of our algorithm. Due to the view-dependent nature of our algorithm, cell sizes remain under a user-specified threshold when the viewport size is changed. Triangles that compose an extracted isosurface are restricted in size, regardless of viewport dimensions. Isosurfaces rendered in a larger viewport are represented by a relatively greater number of triangles and thus obtain greater image quality at the cost of computational time. Figure 12 summarizes the effects of varying the viewport size on the mesh refinement time and the rendering time. The same principle applies when zooming in on a dataset. As the eye moves closer to the dataset, the viewable region of the data set is extracted at a relatively higher resolution, see Figure 14.

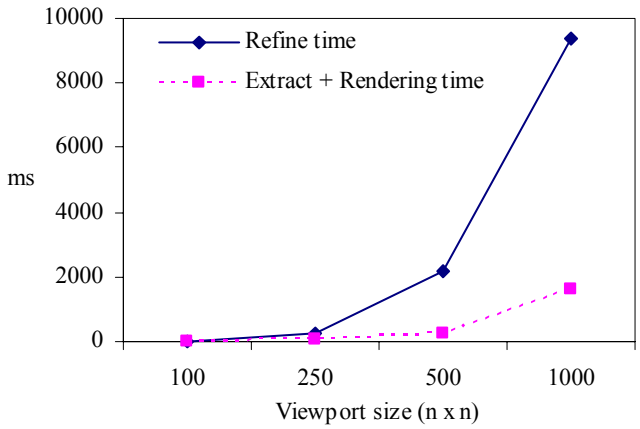
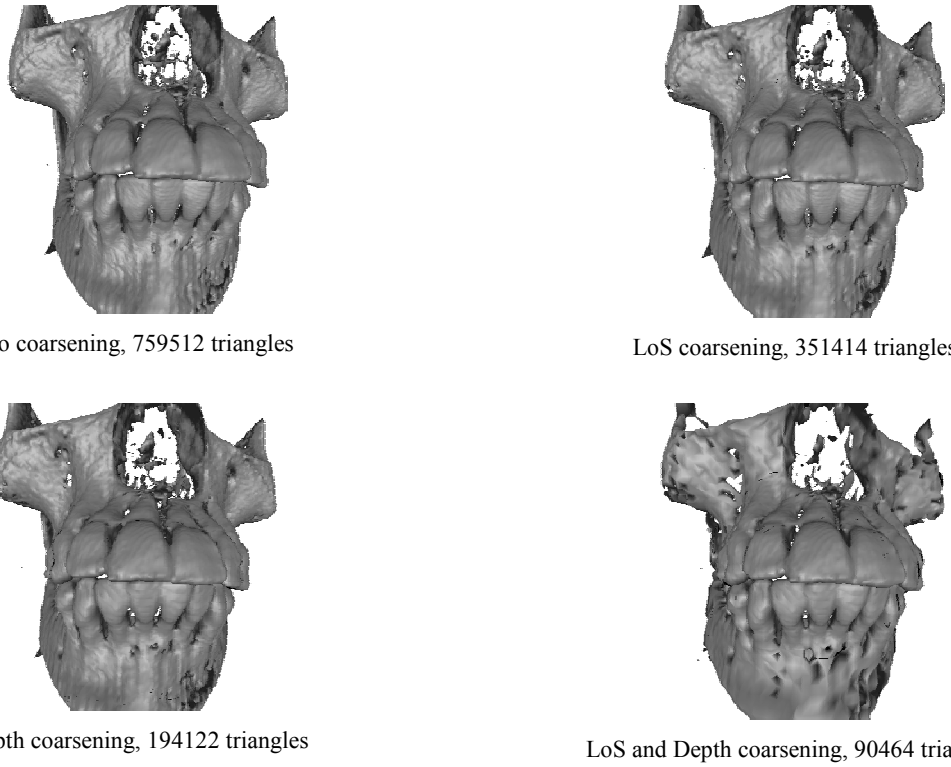


Figure 12. Effects of viewport size on performance for octree 256^3 skull data set, MPPC = 16.

Though our approach is independent of the data set, the results are dependent on the data. Data sets that contain isosurfaces that do not occupy a large number of cells are not particularly suited for our algorithm due to the added overhead of refining the mesh. The advantage of our algorithm becomes apparent when interactively exploring large and complex data sets. The time saved from traversing fewer cells during isosurface extraction offsets the overhead of hierarchical mesh refinement. Larger data sets also have greater depth, allowing our algorithm to take advantage of the effects of perspective viewing.



No coarsening, 759512 triangles

LoS coarsening, 351414 triangles

Depth coarsening, 194122 triangles

LoS and Depth coarsening, 90464 triangles

Figure 13. Effects of coarsening functions on a 256^3 skull dataset for the octree mesh, MPPC = 25. (Dataset courtesy of Siemens Medical Systems, Forchheim, Germany).

The presence of coarsening functions, as well as the magnitude of their effect can be adjusted by the user. Figure 13 demonstrates the effects of different coarsening functions. Note the detail of the front teeth remains relatively high in all the images because they are located close to the eye and near the LoS. Other areas of the skull are reduced to lower resolutions and the region that is affected by each coarsening function can be seen.

62539 Triangles

157403 Triangles

262588 Triangles

Figure 14. Zooming in on a 256^3 bonsai tree using the tetrahedral mesh, MPPC = 25. (Dataset courtesy of S. Roettger, Abteilung Visualisierung und Interaktive Systeme, University of Stuttgart, Germany).

Figure 15 illustrates the results of our view-dependent algorithm for the octree (left) and tetrahedral mesh (right). A full-resolution isosurface extraction is shown on top. At full resolution, the isosurface extracted from the tetrahedral mesh contains a noticeably larger number of triangles than the octree method. Using our view-dependent algorithm, the isosurface extracted from the octree contains fewer triangles than the tetrahedral mesh extraction, but suffers from the presence of cracks. Difference images comparing the full-resolution extraction and our algorithm are shown on the bottom.

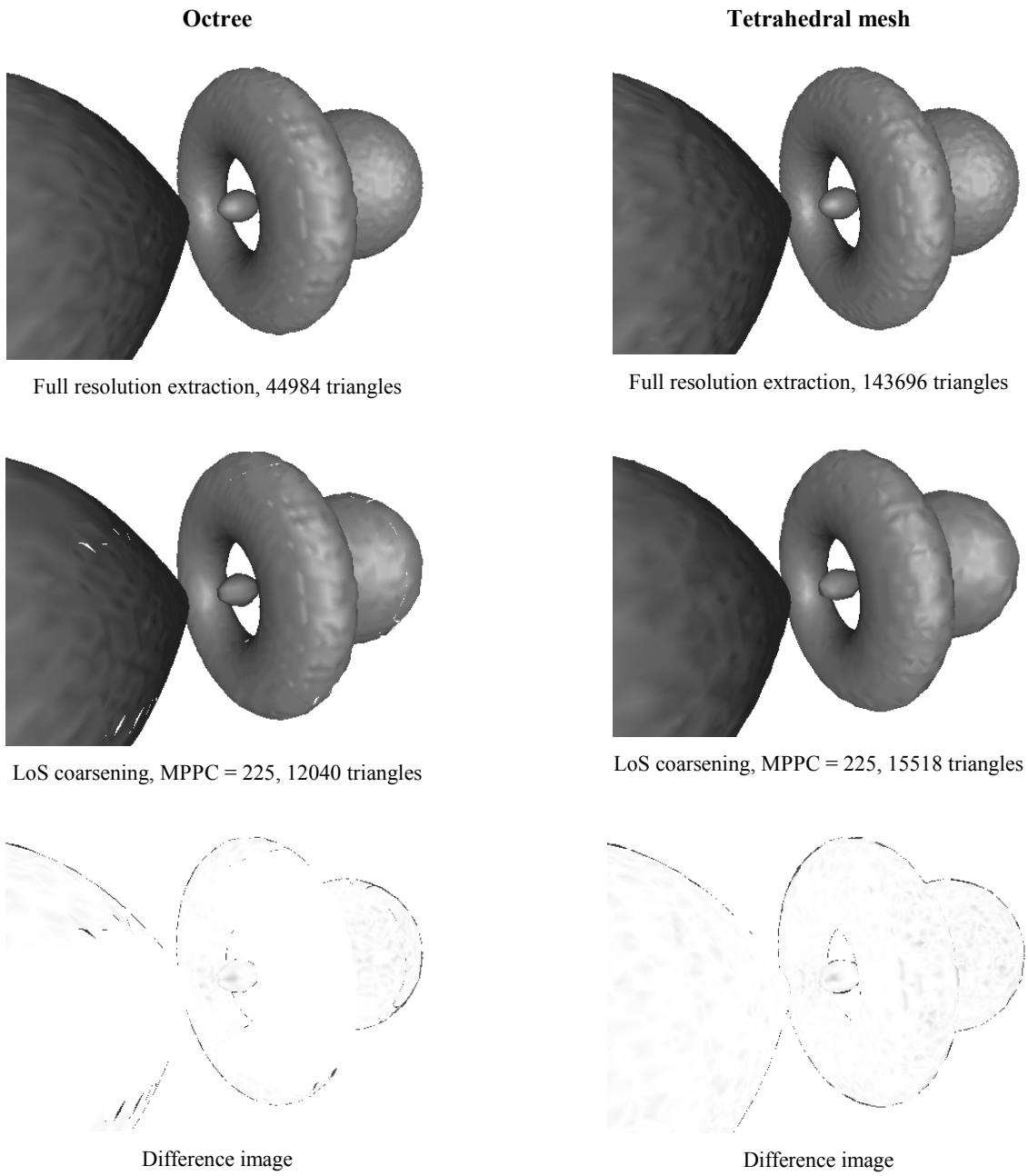


Figure 15. Comparison of full resolution isosurface extraction to view-dependent isosurface extraction using both octree and tetrahedral mesh on a 128^3 hydrogen atom. (Dataset courtesy of the German Research Council)

For a given MPPC, the octree paradigm is better than the tetrahedral mesh in terms of computational time, i.e., time needed to hierarchically refine the mesh, and triangle count. The added costs of the tetrahedral mesh are due to the finer level of granularity caused by the longest-edge subdivision scheme. Finer granularity supports more levels of detail accommodating a higher level of adaptivity. This yields a closer "match" to ideal cell size, subject to the MPPC. Furthermore, while the octree only works for structured-rectilinear grids, the tetrahedral mesh can be used for more general structured-curvilinear grids and even arbitrary grids. Isosurfaces extracted from an octree can suffer from cracks, see Figure 3.

The finer level of granularity leads to a greater number of elements in the mesh and thus increased memory usage when compared to the octree. However, using a more sophisticated encoding method assuming that the data points lie on a $(2n + 1)^3$ grid can be used to store the mesh very efficiently[12]. This technique stores the entire mesh in memory using bit flags to mark the current presence of mesh elements. Memory usage is independent of the refinement level when storing the mesh in such a manner.

7. CONCLUSIONS AND FUTURE WORK

We have presented an algorithm for guiding view-dependent refinement of hierarchical data structures. Our algorithm retains the desired level of detail near areas of interest while maintaining lower resolution in other areas to reduce the amount of data to process. We have implemented and tested our algorithm for isosurface extraction on both an octree and a tetrahedral mesh. The visual quality of our results is near equal to full resolution, at a fraction of the triangle count. The drawback is the amount of time used to refine the view-dependent mesh, but this time can be amortized over several frames with the use of progressive refinement.

We plan to investigate how efficient and near-optimal refinement strategies can be used to improve performance of massive data exploration techniques. Especially, tetrahedral mesh refinement strategies seem to be advantageous, since a computationally and memory-wise efficient implementation can be accomplished by using the concept of a split vertex/diamond. Furthermore, encoding can use bit-shift operations and look-up tables, see Gregorski et al.[12] to improve performance even more. Future work will address a more sophisticated progressive refinement strategy. The goal is to limit the amount of time spent refining a mesh before a frame is drawn and keeping a triangle budget. A triangle cache can be used to store triangles extracted from a mesh element: If a mesh element is present in the next frame, the extraction step can be skipped. Without the need of pre-processing, data-dependent refinement steps can be added, for example, by merging and marking cells found to be empty after initial isosurface extraction step. Extraction and refinement checks can be skipped as long as the isovalue does not change for marked cells, which can improve efficiency even more.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award) and ACI 0222909, 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 National Institute of Mental Health and the National Science Foundation under contract NIMH 2 P20 MH60975-06A2; and the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159. We also 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. W.E. Lorensen, H.E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm", ACM SIGGRAPH Computer Graphics, Vol. 21, No. 4, pp. 163-169, July 1987
2. G.M. Nielson and B. Hamann, "The asymptotic decider: resolving the ambiguity in marching cubes," IEEE in Proceedings of Visualization 91 (G.M. Nielson and L. Rosenblum, eds.), pp. 83-91, 1991

3. A. Gueziec and R. Hummel, “Exploiting triangulated surface extraction using tetrahedral decomposition,” IEEE Trans. Vis. Comp. Graph. 1, pp. 328-342, 1995.
4. S. Chan and E. Purisima, “A new tetrahedral tessellation scheme for isosurface generation,” Computer & Graphics, 22(1):83-90, 1998.
5. P. Cignoni, C. Montani, E. Puppo, and R. Seopigno, “Speeding up isosurface extraction using interval trees,” IEEE Transactions on Visualization and Computer Graphics, 3(2): 158-170, 1997.
6. R. Shekhar, E. Fayyad, R. Yagel, J.F. Cornhill, “Octree-based decimation of marching cubes surfaces”, IEEE in Visualization 96 (R. Yagel and G. M. Nielson, eds.), pp. 335-344
7. C. L. Bajaj, V. Pascucci, D. R. Schikore, "Fast isocontouring for improved interactivity" In 1996 IEEE Symposium on Volume Visualization Proc. (R. Crawfis and C. Hansen, eds.), San Francisco (CA), Oct. 28-29, pp. 39-46, 1996.
8. R. Shu, C. Zhou, M.S. Kankanhalli, “Adaptive marching cubes”, The Visual Computer, Vol. 11, No. 4, pp. 202-217, 1995
9. Y. Zhou, B. Chen, and A. E. 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, Nov. 1997.
10. S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive ray tracing for isosurface rendering," in IEEE Proceedings of Visualization '98, October 1998.
11. Y. Livnat and C. Hansen. “View dependent isosurface extraction,” In Proceedings of IEEE Visualization '98 (D. Ebert, H. Hagen, and H Rushmeier, eds.), pages 175-180, 1998.
12. B. Gregorski, M. A. Duchaineau, P. Lindstrom, V. Pascucci, K. I. Joy, "Interactive View-Dependent Rendering Of Large Isosurfaces", in Proceedings of the IEEE Visualization 2002 (H. Pfister and M. Bailey, eds.), October/November 27-1, 2002.