Multiresolution Techniques for Interactive Texture-based Rendering of Arbitrarily Oriented Cutting Planes

Eric LaMar⋆
Mark A. Duchaineau*, Bernd Hamann*, Kenneth I. Joy*

Center for Image Processing and Integrated Computing
Department of Computer Science
University of California, Davis, CA 95616-8562, USA

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Box 808, L-561 Livermore, CA 94551, USA

Abstract. We present a multiresolution technique for interactive texture based rendering of arbitrarily oriented cutting planes for very large data sets. This method uses an adaptive scheme that renders the data along a cutting plane at different resolutions: higher resolution near the point-of-interest and lower resolution away from the point-of-interest. The algorithm is based on the segmentation of texture space into an octree, where the leaves of the tree define the original data and the internal nodes define lower-resolution versions. Rendering is done adaptively by selecting high-resolution cells close to a center of attention and low-resolution cells away from it. We limit the artifacts introduced by this method by blending between different levels of resolution to produce a smooth image. This technique can be used to produce viewpoint-dependent renderings.

1 Introduction

Computing technology has steadily improved for more than four decades and continues to improve rapidly. These increased computing capabilities have enabled applications to scale accordingly in overall throughput and resulting data set sizes. However, current visualization techniques break down when operating in this environment due to the massive size of the data sets. New techniques are necessary to enable exploration of large multidimensional data sets.

In this paper, we combine hardware-assisted texture mapping and multiresolution methods for rendering cutting planes of large volumetric data sets. The general idea is to assign priorities to different regions of the volume and to render the high-priority regions with highest accuracy, while lower-priority regions are rendered with progressively less accuracy, and progressively faster.

We use an octree to decompose texture space producing several coarser levels of the original data set. Each level is associated with a level in the octree and each level

⋆ eclamar@cipic.ucdavis.edu, duchaine@llnl.gov, {hamann,joy}@cs.ucdavis.edu is half the resolution of the next level. The leaf nodes are associated with the original resolution, and the root node is associated with the coarsest resolution. Interior nodes are created by subsampling the eight child nodes. Each node contains two texture tiles, called *high* and *low*. The *high* tile stores the node's copy of the data; the *low* tile stores portion of the parent's *high* tile that covers the same area as the node.

Rendering a cutting plane involves traversing the octree and applying a selection filter to each node, building a selected node tree. Three results are possible: (1) the node (and its children) are skipped entirely; (2) the node is skipped, but its children are visited; or (3) the node is rendered and the children are skipped. The selected node tree forms an incomplete octree with the leaves being the nodes selected for rendering. The second step is to balance the selected node tree: all adjacent nodes must differ by no more than one level of resolution. The final step is to render each node, blending the *high* and *low* tiles when the node is adjacent to a lower-resolution node.

This technique reduces the amount of data accessed to produce a rendering. This is important in data mining or visual steering applications, where a user does not know the point-of-interest or would just like to browse the data. Another application is progressive visualization: often, a data set is too large to be placed on one computer system, and portions are distributed across a network of machines. It is not always practical to wait for all systems to finish rendering. With our technique, an initial approximation is first rendered. As higher-resolution data is received, a higher-quality approximation is rendered. This continues until all the data is received or the user changes viewing parameters.

Section 2 contains a survey of related work. Section 3 discusses construction of the texture hierarchy, and Section 4 covers how to process and render the texture hierarchy. Section 5 shows results for two data sets and provides performance results. Conclusions and future work are presented in Section 6.

2 Related Work

High-performance computer graphics systems are evolving rapidly. Silicon Graphics, Inc. (SGI) has been a primary developer of rendering technology, introducing the RealityEngine graphics system [1] in 1994 and the InfiniteReality graphics system [9] in 1998. SGI has also provided extensions to OpenGL [10], [8] that allow taking advantage of this hardware.

Cabral et al. [2] show that volume rendering and reconstruction integrals are generalizations of the Radon and inverse Radon transforms. They show that the Radon and inverse Radon transforms have similar mathematical forms and, by developing this relationship, show that both volume rendering and volume reconstruction can be implemented with hardware-accelerated textures. Cullip and Neumann [3] discuss general implementation issues for hardware-assisted texture-based volume visualization and illustrate the superiority of viewport- versus object-aligned sampling planes. Wilson et al. [14] and Van Gelder and Kim [12] develop the mathematical foundation for generating texture coordinates. Van Gelder and Kim also introduce a quantized gradient method for interactive shading for volume visualization. Westermann et al. [13] show how to visualize isosurfaces using fragment testing and discuss a technique to shade the texture-based isosurfaces. Grzeszczuk et al. [5] enumerate many methods using hardware-accelerated texturing to provide interactive volume visualization, and they introduce a library for texture-based rendering called *Volumizer* [4].

LaMar et al. [7] discuss techniques on which this work is based. This paper [7] shows that multiresolution techniques, when applied to large data sets and used for volume rendering applications, are a reasonable approach to reducing both rendering time and amount of data rendered. Shen et al. [11] discuss a temporally based multiresolution scheme for volume visualization of unsteady data sets.

Our method differs from these prior approaches in that we allow adaptive rendering of a cutting plane. Prior algorithms assume that a data set is “uniformly complex” or “uniformly important.” This is not the case in an immersive environment, where data closer to the viewer has more visual importance than data far away. Our method of rendering tiles at different resolutions enables us to treat quality as a “tunable” parameter. Artifacts that may appear are removed by blending higher-resolution nodes into lower-resolution nodes.

3 Generating The Texture Hierarchy

3.1 *High*/*Low* Texture Tiles

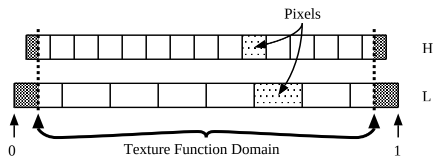

Fig. 1. A node with one-dimensional tiles, *high(H)* and *low(L)*.

In hardware texturing, linear interpolation is used to interpolate the values at the centers of adjacent texels. To allow for blending within a node, each node contains two texture map tiles (Figure 1). The *high* tile is the normal data associated with that node. The *low* tile is that part of the parent’s *high* tile that is covered by the child node. The size ratio *high* to *low* is defined as $|high| = |low| * 2 - 1$. Thus one of the tiles must have odd size. If the size of a texture tile must be a power of two, then this relationship will incur some memory overhead. Our system uses a power-of-two size for the *low* tile, and the size for the *high* tile is calculated accordingly.

3.2 The Multiresolution Texture Hierarchy

Figure 2 shows a texture hierarchy consisting of two levels. The higher-resolution level is denoted as level A, with nodes A^0 and A^1, and the lower-resolution level as B. The image represented by A can be approximated by B. The *high* and *low* tiles in B are the same size as the *high* and *low* tiles in A^0 or A^1, and half the total size of the *high* and

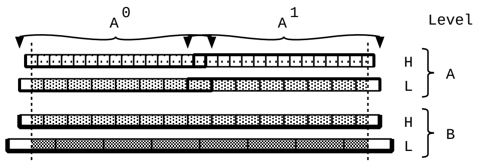

Fig. 2. A texture hierarchy of two levels.

low tiles in A. We note that the natural relationship for two textures whose resolutions differ by a factor of two is using texel-center alignment. In the binary tree arrangement defined by this one-dimensional texture, B is the parent of A^0 and A^1. Also, note the correspondence between the *low* tile of the children to the *high* tile of the parent.

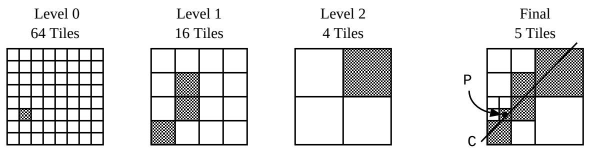

Fig. 3. Selecting a set of tiles from a 2D hierarchy of four levels (level 3 not shown).

Figure 3 shows a two-dimensional quadtree example. The original texture, level 0, contains 64 nodes. The dark regions show the portion of the level used in rendering the cutting plane. Nodes are selected when the distance from the center of the node to the point p is greater than the diagonal length of the node, and when the node intersects the cutting plane c. The selected nodes are shaded. The original texture, divided into 64 nodes, requires 64 time units to transfer. The multiresolution rendering uses five nodes, requiring five time units which implies a speed-up factor of about 13.

This technique extends to three-dimensional textures. Approximations are generated by subsampling the textures. The amount of memory “wasted” over the prior technique [7] is the storage of the *low* tile with each node; since each *low* tile is $\frac{1}{8}$ the size of the *high* tile, the additional memory overhead is $\frac{1}{8}$.

4 Rendering

The rendering phase is divided into the following steps: (1) selecting nodes to be rendered and building the selected node tree; (2) balancing the selected node tree; (3) computing the blending ratios; and (4) rendering the nodes.

4.1 Selecting Nodes

The first rendering step determines which nodes will be rendered. The general filtering logic starts at the root node and performs a depth-first traversal of the octree. For each node, we evaluate a selection filter, which returns one of three possible responses:

– Ignore this node and all of its children. This response is used to cull the tree. For example, if a node is not in the view frustum, then we can ignore the node and its children.
– The node satisfies all criteria. Render the node and do not consider the children.
– The node does not satisfy the criteria. Check the children.

Our primary selection filter is based on one of these two criteria:

– Cutting Plane. This filter selects a node when it intersects the cutting plane.
– Multiresolution Cutting Plane. This filter selects a node when it intersects the cutting plane and the distance from the node center to the point-of-interest (on the cutting plane) is smaller than the diagonal length of the node.

4.2 Blending

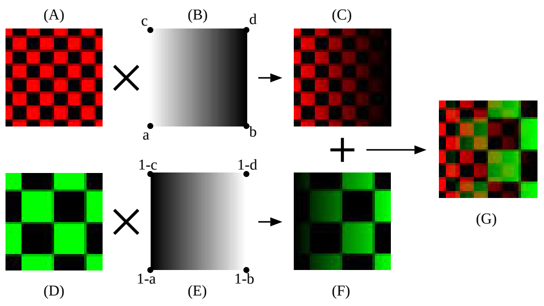

Fig. 4. Blending red and green checker board patterns.

Texturing is performed by modulating the color of the proxy geometry by the texture; the color is white and constant across a polygon. However, to blend two images, we can change the polygon color to implement bilinear filtering. In Figure 4, image (G) is created by performing a per-pixel affine combination of images (A) and (D). Image (B), with ratios of $a = c = 1$ and $b = d = 0$, multiplies (A) and produces (C). Image (E) multiplies (D) and produces (F). Images (B) and (E) sum to unity. Adding (C) and (F) produces (G): a transition from red checks on the left to green checks on the right. We obtain a smooth transition provided (A) and (D) are two different resolutions of the same image.

4.3 Neighborhoods and “Balancing”

The blending algorithm described in section 4.2 requires that all selected nodes in a 26-neighborhood (across node faces, edges, and corners) have resolutions that differ by at most one level in the octree. Blending within a node can only blend between two texture resolutions: the high-resolution texture is blended into the low-resolution texture. Nodes have two textures tiles, *high* and *low*, so that a pair of nodes that differ by one level in the tree can be blended. Those that differ by two or more levels do not share any textures and cannot be blended.

After balancing the tree, we examine the neighbors of all selected nodes. The nodes adjacent to a node of lower-resolution must be blended such that the textures match. For each corner of a given node, when any of the seven adjacent nodes exist and have a lower-resolution, that corner must blend to the *low* tile; otherwise, it must use the *high* tile.

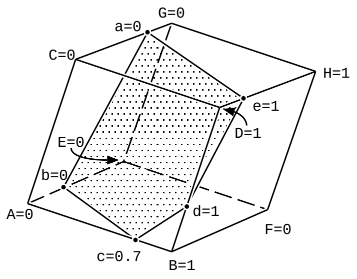

Fig. 5. Cutting plane clipped to an intersecting node.

Figure 5 shows a cutting plane clipped to an intersecting node. *A* to *H* are the blend ratios associated with the node: corners B, E, and H are adjacent to lower-resolution nodes, so that the blend ratio is one; the other corners have a blend ratio of zero, selecting the *low* and *high* tile of the node, respectively. The values *a* to *e* are the blend ratios associated with the clipped cutting planes vertices. Ratios on an edge are linear combinations of the ratios at the ends of that edge, and are proportional to the position of the point along the edge.

For rendering, we first define the RGB value for each clipped cutting plane vertex to the ratio (*a* to *e* in Figure 5), download the *low* texture tile, and draw the polygon. The color values will be interpolated across the polygon, multiplying the texture and producing the first weighted image. Next, we download the *high* texture tile, define the RGB value for each clipped cutting plane vertex to one minus the ratio, and draw the polygon, producing the second weighted image. Finally, by adding the first and second images, we produce the blended result.

5 Results

We have implemented the algorithm and applied it to parts of the Visual Female data set. The data sets were rendered on an SGI Onyx2 computer system with 512MB of main memory and 16MB of texture memory, using a single 195MHz R10K processor.

For comparison, Figure 6 shows a multiresolution image of a Mandrill. This image is used to point out the artifacts when not blending across different levels of resolution. Image 6(b) shows the nodes and node boundaries: the resolution is shown by the node's

	Mandrill (Fig. 6)		Visible Female (Fig. 7)	
Data set resolution	$256^2 * \text{RGB}$ (2D)		$500^2 * 250 * \text{RGBA}$ (3D)	
Data set size	192K		238MB	
Tile resolution (high/low)	$15^2/8^2$		$32^3/16^3$	
Tile size (high/low)	$1024/256$ bytes		$128\text{K}/16\text{K}$ bytes	
Level 0 nodes	324		2601	
Rendered nodes: fixed/MR	324	41	443	50
Bytes transmitted	405K	51K	56MB	7MB
Rendering time	-	-	2.0 sec.	0.37 sec.

Table 1. Timing results for Mandrill and Visible Female data sets.

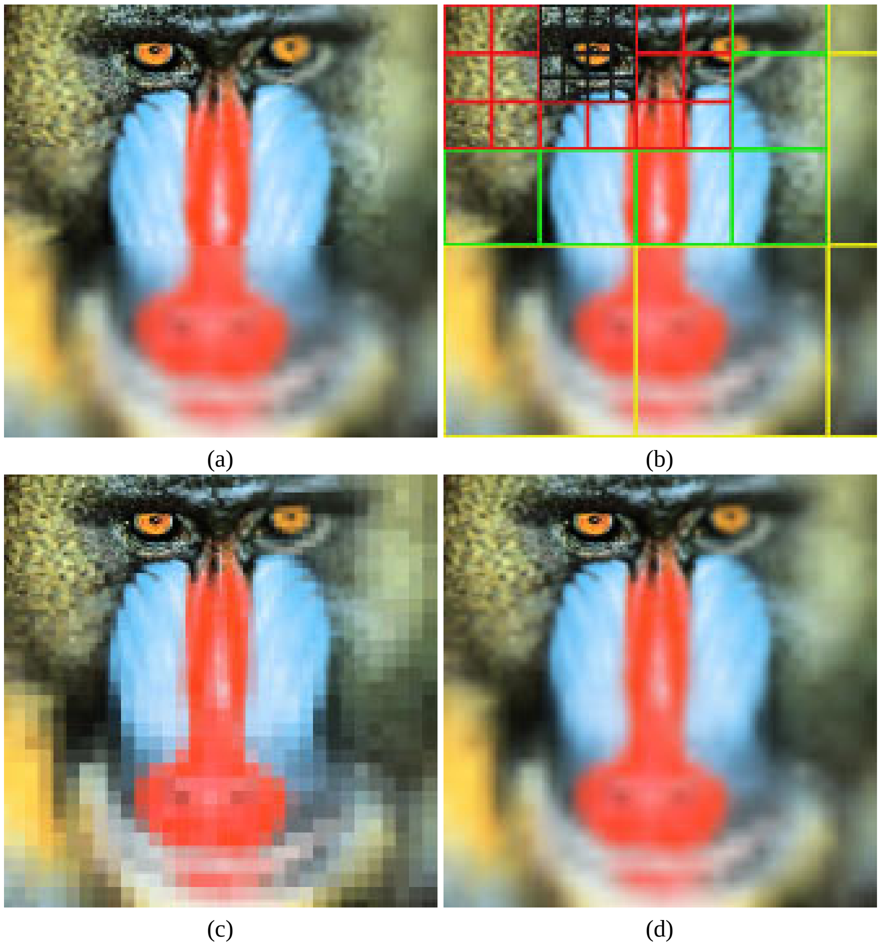

Fig. 6. Multiresolution Mandrill: (a) without blending; (b) node boundaries high-lighted; (c) blended nearest-neighbor; and (d) blended, bilinear filtering.

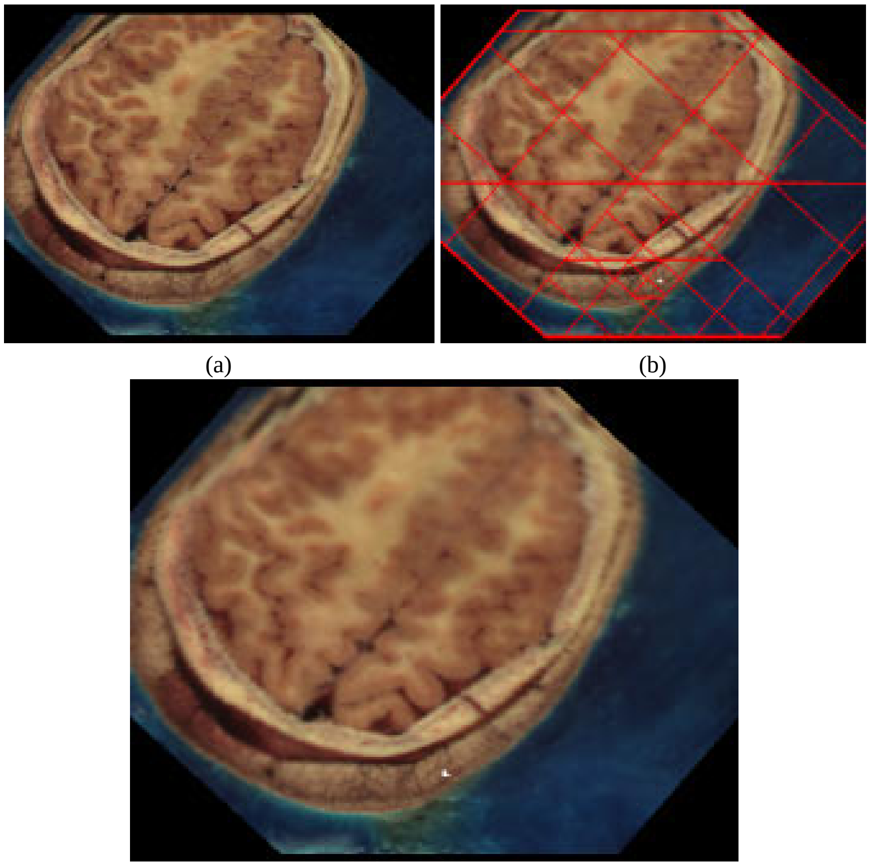

Fig. 7. Multiresolution cutting plane of the Visible Female data set: (a) fixed resolution; (b) blending with node boundaries high-lighted; and (c) MR, blending.

boundary color, from highest to lowest: black, red, green, and yellow; notice the artifacts at the node boundaries in image 6(a). Image 6(c) shows the blending result, with nearest-neighbor filtering; notice that the pixel sizes blend smoothly across the nodes. Image 6(d) shows the final result; notice how the image is free of the boundary artifacts and smoothly blends high resolution nodes to low resolution nodes.

Figure 7 shows a multiresolution view of the Visible Female data set. The 443 nodes of the Visible Female represent the highest-resolution nodes that intersect the cutting plane (the other 2158 are never considered). The performance results shown in Table 1 are for a single frame; at 20 frames per second. The 1.1GB/sec required for the non-multiresolution approach exceeds the SGI InfiniteReality Engine's maximum transfer rate for textures of 320MB/sec by a factor of about 3.5, while the 140MB for the multiresolution approach has capacity to spare. The selection criteria are flexible and under user control. When the bandwidth is very low (e.g., over a modem), even fewer nodes can be selected.

6 Conclusions

We have presented an algorithm for interactive rendering of multiresolution cutting planes. We use hardware-based texturing, multiresolution techniques, and image blending to render a smooth approximation of a cutting plane. We have shown that our algorithm can produce a reasonable approximation while using less data. Despite the fact that our overall system is limited by the amount of available texture memory, the algorithm produces very good results, and we expect that this approach will have a major impact on the exploration of massive volumetric data sets that are currently generated in numerous applications.

Future work includes error analysis. We will implement this technique in our multiresolution volume visualization system and extend it to visualizing vector fields.

7 Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award) and through the National Partnership for Advanced Computational Infrastructure (NPACI);the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of ALSTOM Schilling Robotics, Chevron, Silicon Graphics, Inc. and ST Microelectronics, Inc. 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. Kurt Akeley. RealityEngine graphics. In *Proceedings of Siggraph 93*, pages 109–116. ACM, August 1993.
2. Brian Cabral, Nancy Cam, and Jim Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *1994 Symposium on Volume Visualization*, pages 91–98. ACM, October 1994.
3. Timothy J. Cullip and Ulrich Neumann. Accelerating Volume Reconstruction With 3D Texture Hardware. Technical Report TR93-027, Department of Computer Science, University of North Carolina - Chapel Hill, May 1994.
4. George Eckel. *OpenGL Volumizer Programmer's Guide*. SGI, Inc., 1998.
5. Robert Grzeszczuk, Chris Henn, and Roni Yagel. *SIGGRAPH '98 "Advanced Geometric Techniques for Ray Casting Volumes" course notes*. ACM, July 1998.
6. IEEE. *IEEE Visualization 99*, November 1999.

7. Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Hardware Texturing-based Volume Visualization. In *IEEE Visualization 99* [6], pages 355–361.
8. Tom McReynolds and Davis Blythe. *SIGGRAPH '98 "Advanced Graphics Programming Techniques Using OpenGL" course notes*. ACM, July 1998.
9. John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinite Reality: a Real-Time Graphics System. In *Proceedings of Siggraph 97*, pages 293–302. ACM, August 1997.
10. Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*. SGI, Inc., 1998.
11. Han-Wei Shen and Kwan-Liu Ma. A Fast Volume Rendering Algorithm for Time-Varying Fields Using A Time-Space Partitioning (TSP) Tree. In *IEEE Visualization 99* [6], pages 371–377.
12. Allen Van Gelder and Kwansik Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In *Proceesings of 1996 Volume Visualization Symposium*, pages 23–30. IEEE, October 1996.
13. Rüdiger Westermann and Thomas Ertl. Efficiently Using Graphics Hardware In Volume Rendering Applications. In *Proceedings of Siggraph 98*, pages 169–177. ACM, July 1998.
14. Orion Wilson, Allen Van Gelder, and Jane Wilhelms. Direct Volume Rendering via 3D Textures. Technical Report UCSC-CRL-94-19, University of California, Santa Cruz, June 1994.