Multiresolution BSP Trees Applied to Terrain, Transparency, and General Objects

Charles Wiley
A.T. Campbell, III
Applied Research Laboratories, University of Texas at Austin, P.O.Box 8029, Austin, TX, 78713
Stephen Szygenda
School of Engineering, University of Alabama at Birmingham, Birmingham, AL 35294
Donald Fussell
Department of Computer Science, University of Texas at Austin, Austin, TX, 78712
Fred Hudson
IBM Austin Division, 11400 Burnet Road, Austin, TX 78758

Abstract

We present a system for incorporating multiple level of detail (LOD) models of 3D objects within a single hierarchical data structure. This system was designed for a scientific visualization application involving terrain and volume rendering. Our data structure is a modified Binary Space Partitioning (BSP) tree. We describe how our tree construction and traversal routines may be used with a variety of LOD methods. This is demonstrated with two different LOD methods: a new method specialized for terrain elevation height fields, and an existing method for general objects. Images, timings, and storage data for our implementation are provided.

Keywords: BSP trees, Virtual reality, Real-time graphics, Multiple levels-of-detail.

1 Introduction

This research was motivated by a scientific visualization project in which we were asked to produce an interactive display combining renderings of terrain and volume data. The volume data, derived from simulations of radio frequency (RF) propagation, was to be rendered as transparent color-coded slices, volume densities, and isosurfaces. We were faced with two technical problems: the number of polygonal scene primitives exceeded what our hardware could draw at interactive rates, and we needed to produce a correct rendering combining transparent and opaque objects.

A typical area of interest in our application, one degree of latitude by one degree of longitude, contains an immense amount of elevation data. The commonly available United States Defense Mapping Agency data (nominally 90 meter spacing of data samples at the equator) for such a patch contains approximately 2.8 million triangles. This is far too much data be rendered at the desired interactive frame rate of 1/30 to 1/10 of a second. To increase drawing speed, the number of polygons drawn per frame needed to be reduced without introducing severe visual artifacts.

Transparent renderings of the volume data needed to be overlaid on the terrain to provide location cues and to give a better understanding of the effects of the terrain on the simulation. Combining transparent and opaque objects presented difficulties. The volume data could potentially intersect the terrain, so we could not just draw the terrain and then draw the volume data.

There was no simple solution for the drawing order of the volume data. Alpha blending is a non-commutative operation, so transparent objects must be drawn in strictly back-to-front or front-to-back order. When volume data is organized in a single axis-aligned grid, a visible priority sort can be derived by correctly indexing the data. Unfortunately our data was generated by several simulations over different, overlapping grids. This eliminated most of the simple sorting options, and it presented the potential for intersecting transparent polygons, which are not handled correctly by sorting alone.

This paper describes the algorithms and data structures we developed in solving the problems described above. Our results proved to be applicable beyond the scope of our initial problem to more general visualization systems. We handle multiple LOD representations along with correct rendering of complex transparent polygon sets without needing to explicitly sort the polygons for each frame. Our solution may be used in conjunction with a wide variety of LOD methods.

The remainder of this paper is organized as follows. Section 2 reviews relevant prior work. Section 3 gives an overview of our approach. In Section 4 we define the Multiresolution BSP tree and show how to build it, and in Section 5 we show how to use our new data structure for rendering. In Section 6 we show how other LOD algorithms may be used within our system. Section 7 pro-

vides implementation details and test results, and in Section 8 we present our conclusions and discuss directions for future work.

2 Previous Work

Algorithms for reducing the number of polygons drawn may be broadly classified into *visibility culling* [7, 8] and *level of detail* (LOD) [9] methods. For our application all views of interest are from the exterior, and the majority of scene surfaces are visible at all times. Thus visibility culling is not applicable. Among LOD algorithms are techniques that construct a triangular mesh that closely approximates a terrain surface while minimizing the number of triangles [22, 26, 28], adaptive subdivision to fit a set of polygons to a surface [5], and decimation to remove vertices[7, 24]. Approaches that decide when to use the simplified models include a method to maintain a minimum frame rate while providing the best possible image [7], using distance from the viewpoint as a decision metric [36], and using hybrid metric functions [11, 13]. None of this work has addressed transparent objects in the scene.

Most virtual world or simulation systems use a location-based approach to segmenting the terrain model [11, 13, 20, 36, 37]. Typical data structures are grids and quadtrees. The grid approach has a single level structure based on a single resolution. The quadtree approach is based on a recursive subdivision of space, wherein each area is divided into four quadrants recursively until reaching some minimum resolution. These approaches work well for scenes containing only terrain, but do not handle the addition of arbitrarily-oriented polygons.

Previous volume visualization systems rely on methods that explicitly sort the primitives based on distance from the viewpoint [14, 15, 25, 33] or on ray-tracing [10, 27] or splatting [32, 35, 12]. These are all fine for producing individual images, but none is fast enough for interaction.

Binary Space Partitioning (BSP) trees facilitate quick back-to-front ordering of scene polygons. Developed for visible surface determination [6, 16, 17, 19, 30], they have been used in several systems that use static scenes [1, 2], and in several dynamic virtual environments[3, 4, 31]. Previous work has not addressed the use of multiple LODs within the BSP tree.

3 Overview of Approach

Our approach is based on the BSP tree, which has several advantages over a quadtree or grid. A quadtree can be easily represented by a BSP tree structure, but the converse is not true. While our terrain tree building method could just as easily produce a quadtree, it would limit the objects that could be inserted into it to also be in a quadtree structure. BSP trees are more general. BSP trees also have several properties that facilitate rendering transparent polygonal objects correctly. The building process automatically detects and splits intersecting polygons. More importantly, a modified inorder traversal algorithm can quickly generate the back-to-front drawing order needed to correctly render the transparent polygons and the terrain.

Our algorithm proceeds as follows. We assume that the terrain contains the majority of the potentially visible polygons. First, several LOD models of the terrain are generated. For this purpose we use a new method of our own devising that takes advantage of the special geometry of digital terrain data, but any LOD method may be used. A Multiresolution BSP tree (to be described in Section 4) is then built from these terrain LOD models. The remaining scene objects are merged into the tree. This tree is traversed at run-time and the resolution for any given sub-tree is selected based on a metric function to maintain image quality and interactive frame rates.

4 Multiresolution Binary Space Partitioning Trees

During construction of a standard BSP tree [30], independent sub-spaces are created. Each of these subspaces is represented as a sub-tree. All polygons contained within a sub-space are included within the subtree corresponding to that sub-space.

In a Multiresolution BSP Tree (MRBSP Tree), alternate LOD representations of the objects within a subspace are generated. These representations, containing different numbers of polygons, do not intersect any other sub-space in the tree. Sub-trees are built for each different representation, and pointers to all the alternative subtrees are maintained as illustrated in Figure 1. The subtree to use for rendering is chosen at run-time using a metric based on the projected size of the sub-space. The LOD selected for one sub-tree need not be the same as what is selected for another. A MRBSP tree may be built for any 3D object that can be isolated from the other objects in a BSP tree and for which several alternative polygonal models are available.

4.1 Building the MRBSP Tree for Terrain Elevation Height Fields

The regular sampling structure of 2D field data makes generating a set of LOD models much easier than for a general object. Our implementation builds LODs from the terrain elevation data that have twice as many sample points in both the longitude and latitude direc-

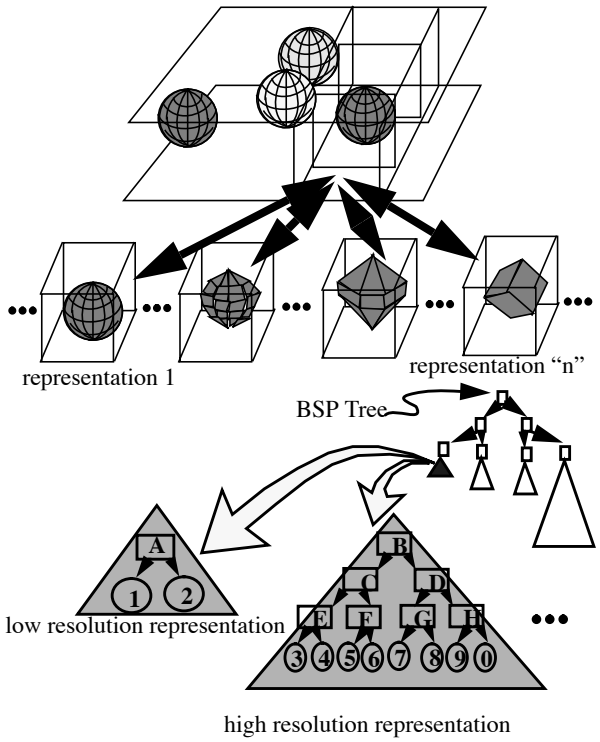


Figure 1: Alternate LOD Example.

tions between levels. Thus the number of polygons increases by a factor of four between each representation level in the tree. Polygons of each size or resolution are grouped into polygon sets before the tree construction process begins. The number of sets used is controllable by the operator and the resolution of the terrain elevation data available.

Splitting planes are chosen along lines of constant longitude and latitude while trying to maintain "square" regions in each sub-space within the tree. These lines are chosen to lie on one of the grid sample lines used to make the polygon sets for each LOD representation so that terrain polygons are not split by these planes. These lines of constant latitude or longitude are chosen from the lowest resolution LOD polygon set currently available, since these planes will also lie along polygon edges for higher resolution LOD polygons. When a level is reached on which the lowest resolution representation, polygon set one, has two sub-child lists with a single rectangular grid (two triangles), a low resolution LOD is created and the remaining resolution polygon sets, sets two through n, are used to create a high resolution LOD. These two LODs are placed into the front and back LOD lists for this splitting plane node. The LOD choosing function determines which LOD to use to render each child region at run-time. The high resolution LOD child is constructed in the same manner recursively until all the LOD polygon sets have been used.

4.2 Merging Other Objects

Once the multiresolution tree representing the terrain is constructed, other objects may be merged into the structure to yield the final scene tree traversed during rendering. Objects may be either polygons or other traditional BSP trees. Merging into the multiresolution tree is the same as for a normal BSP tree [16] with the addition that at nodes with multiple resolution children, a copy of the clipped object must be merged into each resolution sub-tree. The MRBSP tree must be reconstructed whenever any of the objects in the tree move or are modified as does a normal BSP tree.

4.3 Sliding the Tree as the View Position Changes

Using the current terrain tree construction algorithm, the resulting MRBSP tree can be quite large and complex. To reduce the image rendering time, we would like to traverse as small a tree as possible. In addition, we would like to keep the tree small to reduce the physical memory required to hold the terrain representation to a minimum. But, we would still like to be able to use large terrain areas within the targeted simulation environments. One way to accomplish this would be to limit the resolution of the terrain tree, but this would seriously compromise our desire for high image quality. Another approach would be to build a high resolution representation of the terrain only in the area immediately around the current view location. This will help maintain both our goals: a minimum size terrain representation and high resolution. Unfortunately, as the user moves about in the simulation environment, he may run off the edge of this terrain patch.

The current terrain tree building algorithm reduces this problem because it produces a tree that may be "slid" across a large terrain data grid in discrete steps to new positions. Only the splitting plane equations of the nodes and the indices to the data points used for the polygons within the tree need to be updated (Figure 2). The relationship between which polygons are in each region is independent of the terrain tree's root location. The vertex indices of each polygon will change in a predictable manner, as will the splitting plane coefficients. The complex LOD hierarchy and other structures need not be regenerated. This method allows the user to quickly move to a new view location, assuming the elevation data is available. In addition, this type of tree can accommodate changes to the terrain elevation data itself without needing to completely reconstruct the tree.

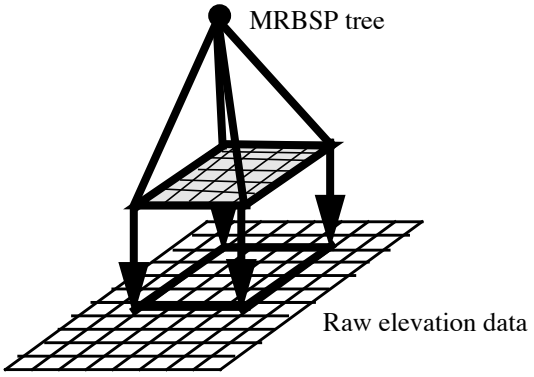


Figure 2: Sliding a tree.

Other objects need to be reinserted into the tree each time the terrain tree is modified. However, if the portion of the terrain over which the objects are located remains in the portion of terrain covered by the updated terrain tree, the object sub-trees may be simply moved over to the new locations in the tree that represent that location in the updated tree. If the object sub-tree moves off the edge of the region represented by the terrain tree, it is merged into the child representing the region off the appropriate edge of the current terrain tree.

5 Rendering the MRBSP Tree

Culling the terrain tree with the viewing frustum greatly reduces the number of polygons to be rendered [17], but the remaining number of polygons still far exceeds the capabilities of current graphics accelerator architectures. To cope with the large number of polygons present in the remaining set, we must further simplify the terrain model and render fewer polygons to represent the terrain where possible. Therefore, the LOD choosing function used by the rendering algorithm needs to balance reducing polygon counts with maintaining image quality. In this implementation, the renderer uses the approximate projected size of the cell on the screen to determine which LOD to use to represent that cell.

Once the tree is created, the LOD to use for each partitioned region within the tree is determined by this function at run-time. Thus, the portions of the tree and the resolutions within each portion used to render each view point image changes dynamically as the viewer moves the camera location. At each visible node for which multiple LODs are available, the function is evaluated for the given region's bounding box and if the result exceeds a threshold, a higher resolution LOD is used, otherwise the low resolution representation is used to render that region.

5.1 Approximate Screen Bounding Box Size Resolution

Our metric for determining which LOD to use is based on the approximate screen coverage area of the three dimensional bounding box of any region in the tree projected to the screen, as shown in Figure 3. The bounding box of a region is determined by the maximum and minimum extents of all of the points from the polygons contained within that region. These values are combined to form an axis aligned bounding box for the region. The bounding box is projected to the screen, and the approximate number of pixels covered calculated. If this screen coverage value exceeds a threshold value, the higher resolution LOD is selected, otherwise the low resolution LOD is used for that region. Since LOD determination is a recursive operation on each subregion within the BSP tree, even higher resolution LODs may be selected for regions for which the high level LOD is selected. Any branch in the tree for which there is a choice of representation has this operation applied to it in order to determine which resolution branch to use in rendering the scene for that particular frame. This function results in lower resolution representations to be used for areas in the scene further from the view point, as illustrated by the overhead view of Figure 4. Figure 5 shows the pseudo code used to render the MRBSP tree.

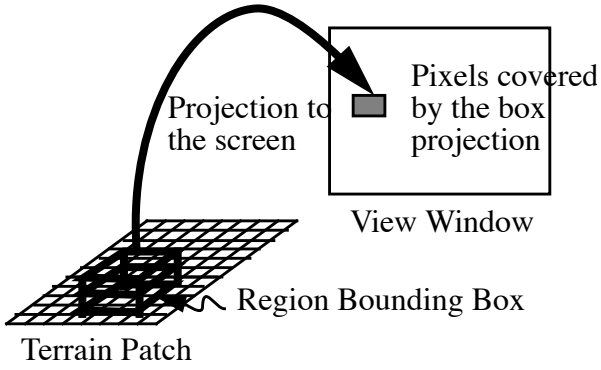


Figure 3: LOD Management Function**.**

6 Using Other Level of Detail Methods

The MRBSP tree may be used with any LOD method. Once a set of LOD models has been generated for a particular object, a MRBSP tree structure can be built from that set. Many methods of producing simplified polygonal models have been reported in the literature. To test our data structure on a general polygonal object, the model generation technique described by Schaufler and Sturzlinger [23] was used to generate multiple LOD rep-

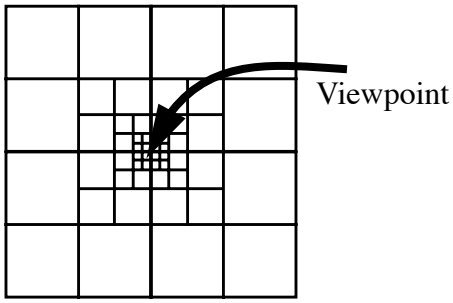


Figure 4: Decreasing LOD resolution with distance from viewer.

resentations for a Volkswagen Beetle. This method uses hierachical clustering to remove vertices from the set used to generate the polygonal representation of an object. LODs generated by this method were used to construct a MRBSP tree using the planes of the maximum bounding volume of the model set to isolate the Volkswagen from any other objects that may also be in the scene. Results from this test are reported in the next section.

```
void draw(BSP *tree, vector3 viewPoint)
{
    int side = evaluateSide(tree->splitPlane, viewPoint);
    int fronLODRep = 0;
    int backLODRep = 0;
    if (tree->numFrontLODs > 1)
        frontLODRep =
            determineLOD(tree->frontLOD[0]);
    if (tree->numBackLODs > 1)
        backLODRep =
            determineLOD(tree->backLOD[0]);
    if (side > 0) {
        draw(tree->backLOD[backLODRep],
              viewPoint);
        drawPrimitives(tree); /* draw the polygons in this node */
        draw(tree->frontLOD[frontLODRep],
              viewPoint);
    }
    else {
        draw(tree->frontLOD[frontLODRep],
            viewPoint);
        drawPrimitives(tree); /* draw the polygons in this node */
        draw(tree->backLOD[backLODRep],
              viewPoint);
    }
}
```

Figure 5: MRBSP drawing function.

7 Implementation and Results

We have implemented the new algorithms and data structured described in this paper with C++, Inventor 1.0, and GL. All the display objects derive from built-in Inventor objects and all data input and output uses functionality provided by Inventor. We used standard Inventor viewers, which give a "through the cockpit" view of the 3D terrain environment. Viewing coordinates may be manipulated by moving the mouse and by using thumbwheels widgets. The system uses Portable Graphics, Inc. ports of these libraries to run on HP PA-Risc based J210 workstations with CRX 48Z graphics adaptors. Texture mapping is done in software. Through experimentation, it was determined that this combination of hardware and software has an approximate limit of 50,000 polygons per second, regardless of rendered image size. This is approximately 1/30 the rendering speed of a SGI RealityEngine, which is the platform for other published work [13].

7.1 Terrain

Figure 6 shows a wire frame view of a terrain patch created with four LODs from digital elevation data for the southern California coastal area. It illustrates the decreasing resolution LODs being used as the BSP tree regions move further from the view point. Figure 7 shows the same terrain patch rendered with a pseudo sun-angle shading texture map applied. Figure 8 shows a more complex isosurface with terrain and figure 9 shows this same isosurface along with an isosurface from another volume along with the terrain.

In comparison tests between full resolution rendering of a terrain patch and using an MRBSP tree to render the same patch, we have seen a rendering speed improvement of 1000 to 4000%. The full resolution implementation uses the triangle set from the highest resolution representation in the multiresolution test. It was implemented using C++ and Inventor, the same as the MRBSP tree implementation, to help normalize the overhead between the two implementations. For the two degree by two degree test patch there are approximately 260,000 polygons in the terrain database. For the full resolution test, the terrain is rendered using z-buffer techniques. The transparent objects are kept in a separate BSP tree structure and are rendered with the z-buffer off after rendering the terrain polygons. Timing data is shown in Table 1. The difference would be greater if the transparent polygons were kept in a simple list needing to be sorted for each view point. Still, the terrain and other opaque objects in the scene needed to be maintained separately from the transparent objects. With the MRBSP tree implementation, the scene database is all merged into a single tree that is traversed to render the images. In testing the terrain build times versus the time to slide an existing tree structure, the time to build a MRBSP tree represen-

Table 1: Comparison of MRBSP Tree to Single Resolution Polygon Implementation.

	Texture Mapped	Wire Frame
MRBSP Tree	0.33 - 0.43 frames/sec.	1.28 - 1.55 frames/ sec.
Full Resolution Polygons	0.0082 - 0.019 frames/ sec.	0.018 - 0.020 frames/ sec.

tation takes approximately 100 times as long as the slide operation.

7.2 General Objects

Table 2 contains the number of vertices and polygons present in each of the individual LOD models of the Volkwagen. The multiresolution partitioning tree model that combines all five of the individual LOD models has a size of 113,517 bytes. The individual model files have a combined size of 103,944 bytes. Thus, the overhead for the multiresolution representation is quite small when compared to the overall size of the data. It is roughly 10,000 bytes. This 10,000 byte overhead or increase in the size of the data for the MRBSP tree divided by the combined size of all the LOD models is approximately 9.6%. Table 3 shows the range of rendering rates recorded when the multiresolution model was viewed from varying distances and spun in the Inventor viewer with spin animation enabled. In each case, the model was placed in a location where the LOD chosen would vary as the Volkswagen spun. The higher frame rates resulted when the LOD with fewer polygons was rendered and the lower rates occurred while the higher resolution LOD model was chosen. Figure 10 shows an individual VW model in wire frame and rendered, as well as a view of the multiple resolution scene containing several VWs rendered using varying LODs.

Table 2: Single Resolution VW Model Rendering Results.

LOD Model	number vertices/ polygons	average wire frame rendering rate	average filled polygon rendering rate
vw0	1147/1078	63.75 f/sec.	12.39 f/sec.
vw3	371/416	140.5 f/sec.	4.8 f/sec.
vw5	247/280	177.7 f/sec.	49.5 f/sec.
vw7	148/169	237.0 f/sec.	72.2 f/sec.
vw9	112/132	263.0 f/sec.	92.8 f/sec.

Table 3: Timing Results of a Single Multiresolution VW, the model file contained 5 LODs

models rendered	Wire frame rendering	Filled polygon rendering
vw0 & vw3	44.38 - 70.78 frames/sec.	5.098 - 15.01 frames/sec.
vw3 & vw5	69.88 - 156.3 frames/sec.	34.89 - 44.17 frames/sec.
vw5 & vw7	85.45 - 106.8 frames/sec.	36.89 - 47.91 frames/sec.
vw7 & vw9	162.8 - 180.1 frames/sec.	35.11 - 58.67 frames/sec.

Finally, a comparison in rendering rates between a scene composed of several MRBSP tree VWs and a scene using just the highest resolution polygons for the same number of VWs was conducted. In this test, eight instances of the VW model were present in the scene database. The MRBSP tree data file produced an approximate rendering rate of 4.5 to 6.5 frames per second. The data file containing eight of the highest resolution polygon sets, implemented as an Inventor indexed face set, was rendered at a rate of approximately 2.2 to 3.0 frames per second. The polygon set was reused for each of the instances by using Inventor's capability to have multiple references to the same object in a scene graph. However, caching of the scene graph was turned off to allow comparison of the complete scene rendering rates as opposed to times to render the display lists stored by GL. Even with the additional overhead of the partitioning tree, the MRBSP tree representation performed better than the full resolution polygon representation by reducing the number of polygons sent to the rendering pipeline.

8 Conclusions and Future Work

This method extends the basic BSP tree construct to facilitate multiresolution. Such representations still produce a correct back-to-front ordering of polygons regardless of the representation traversed, and this ordered polygon list is preserved if parts of the scene tree are traversed at differing resolutions. The LOD branch used to render any portion of this hybrid tree can be selected at run-time and may be adjusted by the user based on system constraints such as minimum acceptable frame rate. By using differing LOD representations, the rendered

image can concentrate most of the processing power on parts of the scene closest to the view location which should contribute the most to any major image features and spend very little of the limited graphics capabilities on parts of the scene far from the view point that contribute to only a small number of pixels on the image. The use of a data structure derived from a BSP tree allows us to inherit all of the attributes of a BSP tree. The tree produces a view point independent back-to-front polygon ordering so the hardware on which the system runs would not need a traditional Z-buffer. This would mean that the target hardware platform would not need RAM dedicated to the Z-buffer nor would it need to do a read-modify-write memory cycle for each pixel produced by the pixel processors in the pipeline for each frame. This should significantly reduce the time needed to process a pixel in the pixel processor stage of the pipeline.

The MRBSP tree structure was specifically applied to the problem of viewing terrain with volume data over it. A method of producing a MRBSP trees of a two dimensional height field was developed. This BSP tree structure allowed insertion of general polygonal objects, rather than being restricted to quadtrees. In addition, the BSP tree splitting plane choosing and construction algorithm provides a structure that can be easily moved without needing to reconstruct the entire tree for any regular rectangularly sampled height field. This allows the in-memory terrain database to be minimized with only a small portion of a very large terrain area needing to be in memory and traversed to render each frame. This should reduce virtual memory page faults caused by having to traverse an extremely large scene database.

Future work will include further improvements to the run-time LOD management function, and modification of the MRBSP tree construction algorithm to allow the resolution of the sets used to construct the tree to vary as a function of distance from the center rather than the current single fixed resolution set. Applications of the MRBSP tree structure to fractal terrain rendering will also be investigated, as will possible extensions to more general objects. Heuristics and methods for merging "good" MRBSP trees also need to be investigated.

9 References

[1] Campbell, A. T., Fussell, D., "Adaptive Mesh Generation for Global Diffuse Illumination," *Computer Graphics*, Proceedings SIGGRAPH 1990, vol. 24, no. 4, pp. 155-164, 1990.

[2] Campbell, A. T., *Modeling Global Diffuse Illumination for Image Synthesis,* Ph.D. Thesis, Department of Computer Science, the University of Texas at Austin, Dec. 1991.

[3] Chrysanthou, Y., Slater, M., "Computing Dynamic Changes to BSP Trees," *EUROGRAPHICS 1992,* vol 11, no. 3, pp. 321-332, 1992.

[4] Chrysanthou, Y., Slater, M., "Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes," *1995 ACM Symposium on Interactive 3D Graphics,* pp. 45-50, 1995.

[5] DeHaemer, M., Zyda, M., "Simplification of Objects Rendered by Polygonal Approximations," *Computers & Graphics,* vol. 15, no. 2, pp. 175-184, 1991.

[6] Fuchs, H., Kedem, Z., and Naylor, B., "On Visible Surface Generation by A Priori Tree Structures," *Computer Graphics,* Proceedings SIGGRAPH 1980, vol. 14, no. 3, pp. 124-133, 1980.

[7] Funkhouser, T., Sequin, C., "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," *Computer Graphics, Proceedings SIGGRAPH 1993*, pp. 247-254, 1993.

[8] Greene, N., Kass, M., Miller, G., "Hierarchical Z-Buffer Visibility," *Computer Graphics, Proceedings SIGGRAPH 1993*, pp. 231-236, 1993.

[9] Heckbert, P., Garland, M., "Multiresolution Modeling for Fast Rendering," *Proceedings of Graphics Interface 1994,* pp. 43-50, 1994.

[10] Kajiya, J., Kay, T., "Rendering Fur with Three Dimensional Textures," *Computer Graphics,* Proceedings SIGGRAPH 1989, vol. 23, no. 3, pp. 165-174, 1989.

[11] Koller, D., Lindstrom, P., Ribarsky, W., Hodges, L., Faust, N., Turner, G., "Virtual GIS: A Real-Time 3D Geographic Information System," Graphics, Visualization and Usability Center Georgia Institute of Technology Tech Report 95-14, 1995.

[12] Laur, D., Hanrahan, P., "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics,* Proceedings SIGGRAPH 1991, vol. 25, no. 4, pp. 285-288, 1991.

[13] Lindstrom, P., Koller, D., Hodges, L., Ribarsky, W., Faust, N., Turner, G., "Level-of-Detail Management for Real Time Rendering of Phototextured Terrain," Graphics, Visualization and Usability Center Georgia Institute of Technology Tech Report GIT-GVU-95-06, 1995

[14] Lorensen, W., Cline, H. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm,", *Computer Graphics*, Proceedings SIGGRAPH 1987, vol. 21 no. 4, 1987.

[15] Max, N., Hanrahan, P., Crawfis, R., "Area and Volume Coherence for Efficient Visualization of 3-D Scaler Functions," *Computer Graphics,* vol. 24, no. 5, pp. 27-33, 1990.

[16] Naylor, B., Amanatides, J., Thibault, W., "Merging BSP Trees Yields Polyhedral Set Operations," *Computer Graphics*, Proceedings SIGGRAPH 1990, vol. 24, no. 4, pp. 115-124, 1990.

[17] Naylor, B., "Interactive Solid Geometry Via Partitioning Trees," *Graphics Interface '92*, pp. 11-18, 1992.

[18] Naylor, B., "Constructing Good Partitioning Trees," *Graphics Interface 1993*, pp. 181-191, 1993.

[19] Paterson, M., Yao, F., "Binary Partitions with Applications to Hidden-Surface Removal and Solid Modelling," *Proceedings of the Fifth Annual Symposium on Computational Geometry,* June 5-7, 1989.

[20] Pratt, D., *A Software Architecture for the Construction and Management of Real-Time Virtual World,* Dissertation, Naval Postgraduate School, Monterey, CA, June 1993.

[21] Rossignac, J., Novak, M., "Research Issues in Model-based Visualization of Complex Data Sets," *IEEE Computer Graphics and Applications,* vol. 14, no. 2, pp. 83-85, 1994.

[22] Scarlatos, L., Pavlidis, T., "Hierarchical Triangulation Using Cartographic Coherence," *CVGIP: Graphical Models and Image Processing,* vol. 54, no. 2, pp. 147-161, 1992.

[23] Schaufler, G., Sturzlinger, W., "Generating Multiple Levels of Detail for Polygonal Geometry Models," *Virtual Environments '95*, pp. 53-62, 1995.

[24] Schroder, W., Zarge, J., Lorenson, W., "Decimation of Triangle Meshes," *Computer Graphics,* vol. 26, no. 2, pp. 65-70, 1992.

[25] Shirley, P., Tuchman, A., "A Polygonal Approach to Direct Scalar Volume Rendering," *Computer Graphics,* vol. 24, no. 5, pp. 63-70, 1990.

[26] Silva, C., Mitchell, J., Kaufman, A., "Automatic Generation of Triangular Irregular Networks Using Greedy Cuts," *Visualization '95,* pp. 201-208, 1995

[27] Subramanian, K., Fussell, D., "Applying Space Subdivision Techniques to Volume Rendering," *Visualization '90*, pp. 150-159, 1990.

[28] Suter, M., Nuesch, D., "Automated Generation of Visual Simulation Databases Using Remote Sensing and GIS," *Visualization '95,* pp. 86-93, 1995.

[29] Teller, S., Sequin, C., "Visibility Preprocessing For Interactive Walkthroughs", *Computer Graphics,* Proceedings SIGGRAPH 1991, vol. 25, No. 4, pp. 61-69, 1991.

[30] Thibault, W., Naylor, B., "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *Computer Graphics*, Proceedings SIGGRAPH, vol. 21, no. 4, pp. 153-162, 1987.

[31] Torres, E., "Optimization of the Binary Space Partition Algorithm (BSP) for the Visualization of Dynamic Scenes," *EUROGRAPHICS 1990*, pp 507-518, 1990.

[32] Westover, L., "Footprint Evaluation for Volume Rendering," *Computer Graphics*, Proceedings SIGGRAPH, vol. 24, no. 4, pp. 367-376, 1990.

[33] Williams, P., "Visibility Ordering of Meshed Polyhedra," *ACM Transactions on Graphics,* vol. 11, no. 2, pp.103-126., 1992.

[34] Yagel, R., "Volume Viewing: State of the Art Survey," *ACM SIGGRAPH '93 Volume Visualization Course Notes,* 1993.

[35] Yoo, T., Neumann, H., Fuchs, S., Pizer, T., Rhoades, J., Whitaker, R., "Direct Visualization of Volume Data," *IEEE Computer Graphics and Applications*, vol. 12, no. 4, pp. 63-71, 1992.

[36] Zyda, M., Pratt, D., Falby, J., Mackey, R., "NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation," *Computers & Graphics,* vol. 17, no. 1, pp. 65-69, 1993.

[37] Zyda, M., Pratt, D., Monahan, J., Wilson, K., "NPSNET: Constructing a 3D Virtual World," *Computer Graphics*, 1992 Symposium on Interactive 3D Graphics, March 1992, pp. 147-156, 1992.

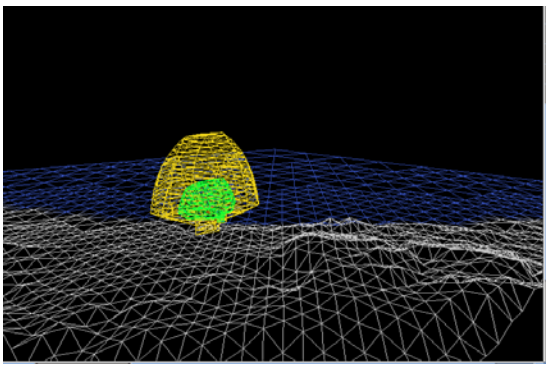


Figure 6: Wireframe Terrain.

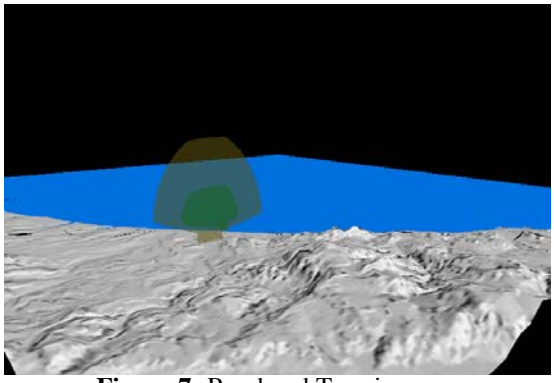


Figure 7: Rendered Terrain.

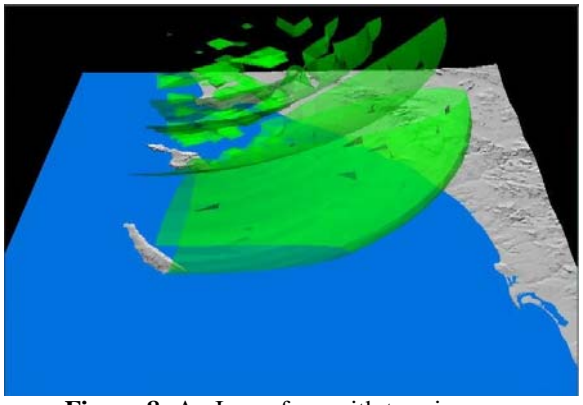


Figure 8: An Isosurface with terrain.

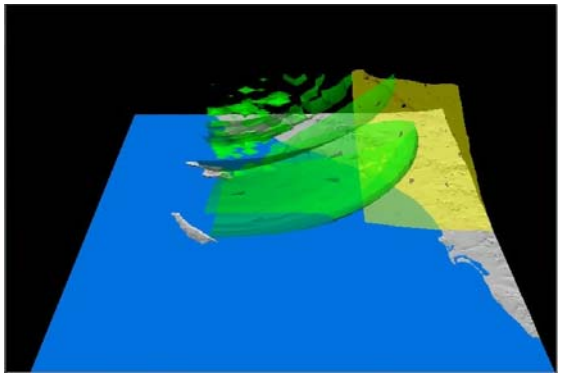
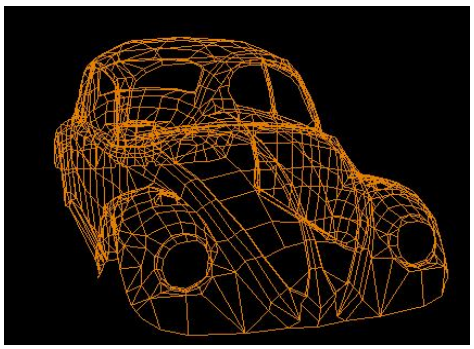
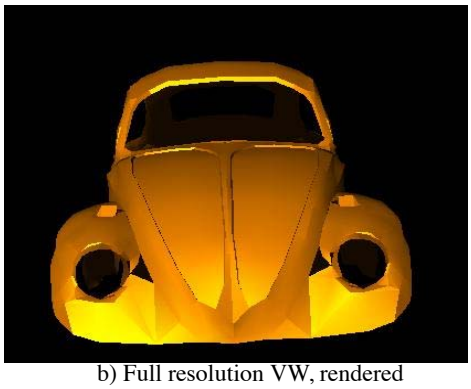


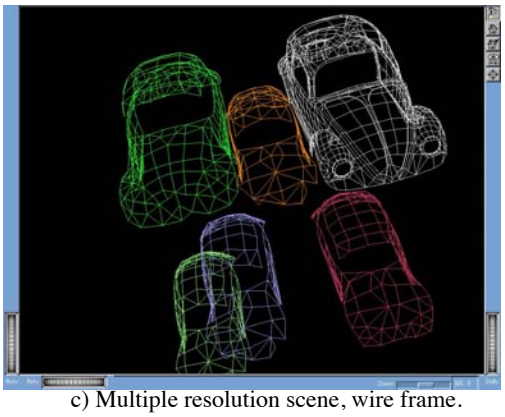
Figure 9: Isosurfaces from two intersecting volumes.



a) Full resolution VW, wire frame



b) Full resolution VW, rendered



c) Multiple resolution scene, wire frame.

d) Multiple resolution scene, rendered.

Figure 10: Volkswagen Beetle Images.