CONSTRUCTION AND RENDERING OF MULTIRESOLUTION BINARY SPACE PARTITIONING TREES.

Joaquín HUERTA, Miguel CHOVER, José RIBELLES, Ricardo QUIRÓS
Computer Science Department, Jaume I University
Campus de Penyeta Roja, 12071, Castellón, SPAIN
Tel: (+34) 64 345771 / Fax: (+34) 64 345848
huerta@inf.uji.es

ABSTRACT

Space partitioning techniques are a useful means of organizing geometric models into data structures. Such data structures provide easy and efficient access to a wide range of computer graphics and visualization applications like real-time rendering of large data bases, collision detection, point classification, etc. Binary Space Partitioning (BSP) trees are one of the most successful space partitioning techniques, since they allow both object modeling and classification in one single structure. Also, due to the fact that complexity of 3D models is increasing far more rapidly than the performance of graphics system, there is an increasing need for multiresolution modeling techniques. This paper presents a novel method that extends BSP trees to provide such a representation. The models we present have the advantages of both BSP trees and multiresolution representations. Nodes near the root of the BSP tree store coarser versions of the geometry, while leaf nodes provide finer details of the representation. The goal of this work is to build a single tree that provides a high number (nearly a continuous range) of representations of an object at different resolutions, with minimum redundancy. We present algorithms to construct and render multiresolution BSP trees of objects in 2D. We also discuss possible extension of our method to 3D.

Key Words: BSP trees, space partitioning, multiresolution, level of detail, polygon simplification.

INTRODUCTION

One of the main problems of computer graphics applications is how to render complex geometric scenes at interactive rates. Traditional modeling techniques are very inefficient when it comes to storing, transferring, and rendering such complex models. A recent solution to this problem stores for a single object a small number of representations with different accuracy or level-of-detail (LOD). Then, objects further from the viewer are retrieved, transferred and rendered at their coarsest LOD. Conversely, objects near the viewer are retrieved, transferred and rendered at their finest LOD. Their main advantage is that only the necessary geometric detail is used for rendering each object. Furthermore, they provide fast data access by exploiting spatial indexing and hierarchical processing. Speedup is then achieved by propagating information across different LODs.

An evolution of a LOD representation is called multiresolution representations [Heckbert-Garland94] and supports the storage of a large number of representations at different LOD. Resolution of each representation may be constant [Hoppe96] and recently some approaches with variable or view dependent resolution have been developed [Hoppe97].

Our goal is to construct an efficient model supporting multiresolution representation with variable resolution. To achieve it, we focus on certain modeling techniques, like space partitioning, that allow more efficient representations than traditional techniques. The best example is Binary Space Partitioning (BSP) trees [Fuchs et al.80] where the partitioning planes are chosen to be those defined by the polygons of the model. This allows both easier manipulation and faster rendering of the polygon data in the model.

This work combines the advantages of partitioning techniques and multiresolution representations by introducing multiresolution BSP trees (MRBSP tree for short). Our representation provides fast spatial classification due to its BSP tree organization, and LOD rendering control using its multiresolution properties. Tree nodes near the root provide coarser representation of the model, while nodes further down the hierarchy provide more accurate LOD representations of the model. This extension of BSP trees allows efficient modeling and rendering of complex environments, and we expect it to be widely used in computer graphics applications.

Next section introduces the problem and objectives of this paper. Section *Preliminaries* describes some preliminary issues that are necessary before going on to the *Construction* section. Next there is a *Rendering* section were we explain the rendering method whose results are show in section *Results*. Last section presents our conclusions and several ideas for future work.

MULTIRESOLUTION BSP TREES

Storage of multiresolution models requires data structures that allow retrieval of LOD representations according to eye location and view orientation. Several data structures have been proposed for this purpose in the literature: image pyramids [Williams83], volume pyramids [Sakas-Gerth91], textures and reflectance [Perlin84] and polygonal models. In [Puppo-Scopigno 97] a survey on multiresolution models can be found.

Polygonal models are best suited for multiresolution representations. They are simpler and more versatile than any other geometric representation. However, they require simplification, the process of obtaining coarser LOD representations from the original full-detail model. Simplification is controlled by a function that minimizes both the number of polygons of the representation, and the error incurred by the approximation. The main problem of simplification algorithms is the definition of this function. In [Heckbert-Garland97] a survey on simplification algorithms can be found.

The main contribution of our work is the use of BSP trees to store the geometry of a multiresolution model. The MRBSP tree structure allows for fast retrieval of LOD representations, fast rendering and progressive transmission by storing coarser LODs near the root of the tree, and finer LODs near the leaves of the tree. So, given an error threshold, the tree can be pruned discarding those nodes whose measured error is below the threshold. The representation is illustrated in the following figures.

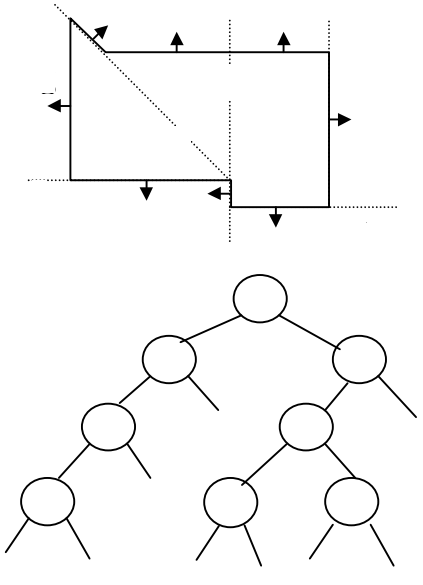


Figure 1:BSP trees for polygon representation. Top: polygon and planes defining boundaries. Bottom: the associated BSP tree representation.

Figure 1 shows a conventional BSP tree used for polygon representation. We can represent

the polygon in that figure by the alternative tree in figure 2. The new tree is multiresolution since nodes *a, b, c,* and *d* provide a coarse representation, while each of the remaining nodes adds a certain amount of detail to the object. *Construction* section describes how to construct such trees in 2D.

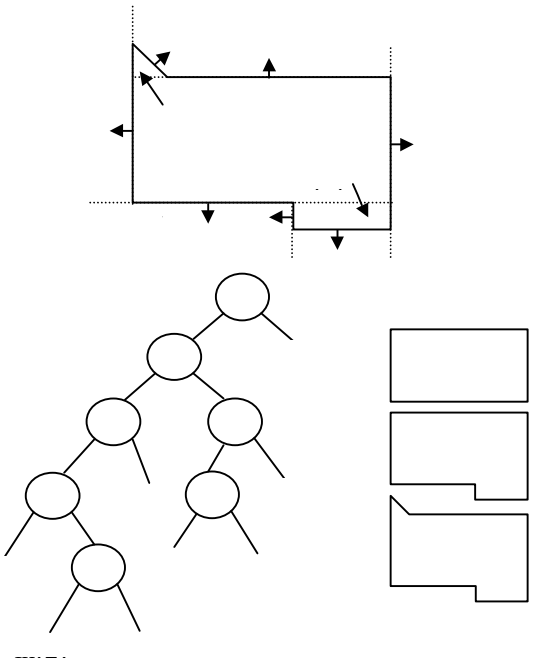


Figure 2. MRBSP tree polygon representation.
Top: polygon and boundary planes.
Left: MRBSP tree representation.
Right: sequence of approximations.

Given a polygon, we want to construct its BSP tree representation by choosing partitioning planes along its edges. The final representation should then contain all the edges of the polygon, plus some others that contribute to our multiresolution purposes. Our objective is to choose planes, or, equivalently, edges in such an order that adding them to the representation increases the amount of detail. This is achieved in two steps. First, we make an initial approximation to the polygon. Then, we choose edges in an order such that the amount of detail added by each choice is maximized. Finally, the algorithm concludes when all edges have been chosen.

In some cases we also choose to include planes that do not correspond to specific edges. The most common case is planes that decimate edges of the original polygon, as the result of an approximation process. Such cases will be described in *Construction* section. Another case arises when modeling concave polygons. Recall that BSP trees are best suited for modeling convex polygons. When a concave polygon is to be represented it may be necessary to divide it into less concave parts, as described in [Huerta et al.97]. This requires adding extra partitioning planes to the representation.

PRELIMINARIES

To construct a MRBSP tree, its is necessary to compute a sequence of approximations to the original polygon. For each approximation we consider two parameters: error incurred by the approximation, and number of cuts in the original polygon. By cuts we mean intersection points between edges of the original polygon and edges of the approximation. Whenever a new partitioning plane is added along an approximation edge, some edges of the polygon may need to be cut in half. Such cuts increase the number of edges and thus the size of the MRBSP tree. Our plane selection algorithm will try to minimize both the error and the number of cuts in the representation [Huerta et al.97]. The sequence of approximations will then be stored in a data structure for later rendering.

Approximation Computation

Among the approximations methods proposed in [Huerta et al.97] we choose the scaling method. It is based on the idea that objects further from the viewer look smaller than objects closer to the viewer. This implies that finer details of a given object disappear as it moves away from the viewpoint, but its overall shape remains unchanged with viewing distance. We can thus use this to compute different LOD representations of an object.

The procedure consists of three steps. During the first step we scale down the polygon by a given scale factor. Then we scan convert each of its edges and determine whether adjacent edges are colinear or not. Finally, if two adjacent edges are colinear, we substitute them by a new single edge.

This procedure is repeated for decreasing scale factors until there are only three edges remaining or it is not possible to remove colinear edges. In order to determine whether two adjacent edges are colinear, we first scan convert both edges separately. Then we scan convert an imaginary edge joining the first vertex of the first edge with the last vertex of the second edge. If the results of both scan conversions coincide, then the new edge can substitute for the two old edges.

Approximation Error

Whenever an approximation is made, it is important to evaluate the quality of the new representation by giving a quantitative estimate of its approximation error. Unfortunately, there are no good error measures, since error depends on visual perception. Usually, measures based on distances, either L_2 or L_∞, are used depending on the application. These measures may be local, global or based in other non-spatial criteria [Puppo-Scopigno 97].

Our own method, as presented in [Huerta et al.97], is a global method based on areas instead of distances. It is motivated by the fact that area-based error evaluation provides better visual results than distance-based errors.

We use an error function that computes the area error as the area difference between the polygon S and the approximation triangle R. That error function is:

$$\hat{D}(R,S) = Area(R-S) + Area(S-R)$$

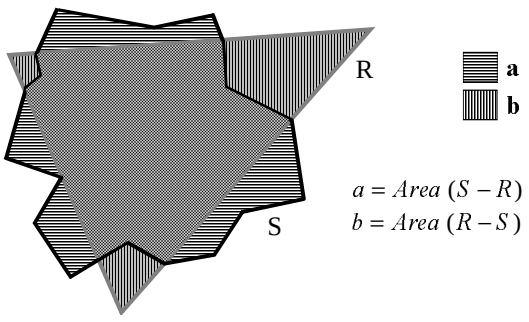


Figure 3. Approximation error.

Data Structures

Our representation is based on a tree structure, where each node stores the plane equation and a list of edges along that plane as well as the customary pointers to its parent and children. Because of the multiresolution feature, it is necessary to store the various edges as seen from different distances along each node's plane. Also we provide a tool to show/hide appropriate edges given a viewing distance. To make this possible, we associate to the list of edges a list of error ranges that indicates when each edge is visible. These ranges allow us to decide which edges to render at any given time as will be described in *Rendering* section.

Specifically, we use the following structures:

Vertex: two fields for the x and y coordinates.
Edge: pointers to its endpoints and lower and upper error bounds where the edge is visible.
Approximation: contains a pointer to the list of vertices of the approximation. Also, it stores approximation error.
Vertex sequence: pointer to list of vertices that are in the same region of the MRBSP tree. It also stores a pointer to the node that has generated it and the value of its area.
Node structure: (figure 4)

Plane equation: A and B (y =Ax + B)
Lowest error for which this node is visible.
Highest error for which this node is visible.
List of edges in the node.
Pointer to parent node.
Pointer to front-child node.
Pointer to back-child node.

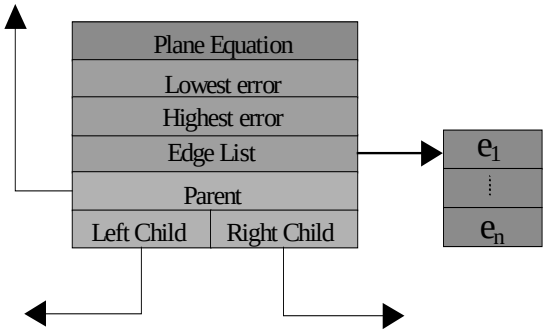


Figure 4. Node structure.

TREE CONSTRUCTION

To add the multiresolution feature to a BSP tree (MRBSP tree), different representations of a single object must be handled. [Wiley et al.97] presents a solution where, given a set of LOD representations of an object, a different tree is constructed for each representation. Then all of these trees are merged into a single one. This solution has several disadvantages:

- Each representation is stored independently, so, there are data redundancies that reduce the amount of storage available for the different LODs.
- The low number of LODs allowed causes popping effects between consecutive LOD, and a poor chance of adapting resolution to viewer distance.

Our goal is to build a single tree that contains a continuum of representations of an object with different resolutions and minimum redundancy. The number of representations is given by size of the MRBSP tree. For the two-dimensional case, trees are of size $O(n \log n)$ [Paterson-Yao90] where n is the number of edges. This provides with a high possibility of adapting the resolution to the different observation conditions. Because changes between resolutions are smooth, it also reduces the popping effect.

The procedure to construct the MRBSP tree is the following: First, using the approximation method presented in *Preliminaries* section, we compute an initial approximation (usually a triangle) of the original object. The approximation process can be stopped when the approximation error is bigger than a certain value, regardless of the number of sides. This error (*Initial_error*) is computed as described in *Preliminaries* section.

Then, a new tree is created with a number of nodes equal to number of sides of the initial approximation. Each of these nodes contains one edge and its corresponding plane coefficients. Initially, the range where each of these edges are visible (error range) is set to [0.0-1.0]. This means that these edges will be displayed always, but this range will be changed as described later. To determine the error range for subsequent edges we use the parameter *Current_error*, computed as:

$$Current_error = \frac{Initial_error}{Polygon\ \ Area}$$

Before proceeding with tree construction, should planes already inserted in the tree cut any of the edges of the original polygon, then new vertices are added at the intersection points in the vertex list. Next, the vertex list is classified into sequences that fall in the same region of the tree.

The procedure is recursive, considering each sequences as a polygon for subsequent recursions. So, we apply the approximation and error computation methods to each one of these sequences. The resulting approximations are then added to the tree sorted by decreasing area. The sequence list is sorted by area and its first element is added to the tree inserting two new nodes. Error range for the edges of these two new nodes is set to [0.0-*Current_error*].

After inserting a new pair of planes in the tree, it is necessary to update their parent node, in order to avoid that an edge of the parent node be rendered when the edges of its children that decimate it, are also rendered (See figure 5).

We name *parent_edge* the edge that is in the parent node and is affected by the insertion of new planes. It will be necessary to avoid rendering any parent edge at the same time that its *children edges* (figure 5 edge a). So, we change *parent edge* error range low boundary to *Current_error,* the same value that *children edges* error range high boundary.

Should part of *parent_edge* must be visible at the same time that its *children edges,* we create one (figure 5 edge b) or two new edges (figure 5 edge c) that represent this visible parts. Error range for the new plane(s) is set the same as *children edges* ([0.0-*Current_error*]). That way, at each node we store all the possibly visible edges with their corresponding error ranges. The error range will be used at rendering time to determine for any viewing parameters, which edges to render. This will be described in *Rendering* section.

Finally, we classify the vertices of the chosen sequence into new sequences with respect to the planes just added, obtaining a new list of sequences of vertices. Again the approximation method is applied to these sequences and its results are included in the sorted list. The process is repeated until the list of approximations became empty as described by the algorithm in figure 6.

Current_error is updated at each iteration with the contribution of the sequence just included (*Error_contrib*$_i$). Using the following notation:

Sequence$_i$: the ith sequence of vertices obtained after classification process.
Approx$_i$: approximation obtained for *Sequence*$_i$.
Error(Approx$_i$*)*: error obtained for *Approx*$_i$.
Area(Sequence$_i$*)*: area of *Sequence*$_i$.

Then:

$$Error_contrib_i = Area(Sequence_i) - Error(Approx_i)$$

$$Current_error = Current_error - \frac{Error_contrib_i}{Polygon\ \ Area}$$

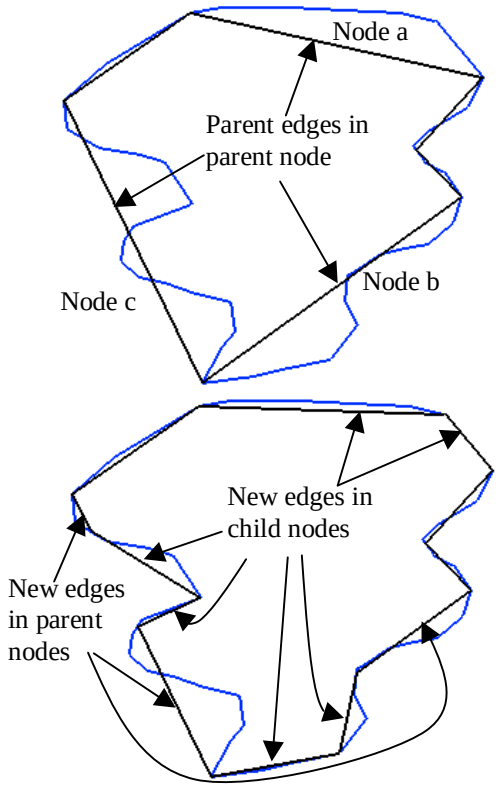


Figure 5. Adding new edges.

```
Algorithm built_mrbspt(p:polygon, t:mrbspt)

    initialize vertex_list with vertices from p
    compute initial approximation a0 of p
    compute intersection of a0 and edges of p and add it to vertex_list
    add nodes to t with planes along a0 edges
    classify vertex_list into vertex_sequence_list in the same region of t
    repeat
        for each item in vertex_sequence_list
            compute approximation ai of vertex_sequence_list[i]
            add approximation ai to approximation_list sort by area
        choose first item of approximation_list aj
        compute intersection of aj and edges of p in the same ragion of t
        and add it to vertex_list
        add two nodes to t with planes along edges of aj
        update error range of parent edge in parent node
        if edges of aj do not span parent edge
            add new edges to parent node
        classify vertex_sequence_list[i] into vertex_sequence_list in the
        same region of t
    until (approximation_list = ∅)
end built_mrbspt
```

Figure 6. MRBSP tree construction algorithm.

```
Algorithm render_mrbspt(v:view_point, t:mrbspt, h:threshold)

   if not_null(t)
      if (approx_error(root[t])<h)
         if pos_side_of (root[t],v)
            render_mrbspt(v, front_branch[t], h);
            render_edge(root[t],h);
            render_mrbspt(eye, back_branch[t], h);
         else
            render_mrbspt(v, back_branch[t], h);
            render_edge(root[t],h);
            render_mrbspt(v, front_branch[t], h);
end display_mrbspt
```

Figure 7. MRBSP tree rendering algorithm.

RENDERING

In this work we consider 2½ D models represented as 2D polygons parallel to the XY plane and are extruded from position Z=0 to their height. Edges are rendered as two triangles forming a quadrilateral perpendicular to the XY plane from (x_0, y_0, 0) to (x_1, y_1, height). For efficiency, rendering is done in a front to back order using the "dynamic screen" structure [Gordon-Chen91].

To decide which edges (sides) to render, we use a simple method that determines the acceptable error (*threshold*) for the current view as a percentage of the window occupied by the projection of the bounding box of the object. The procedure is similar to [Gordon-Chen91] but it prunes those nodes of the tree whose error is lower than the given threshold.

The method proceeds by classifying the view-point in the tree, starting at the root downwards, and rendering the edges (sides) in front of it, then those in the node and finally those behind it. Only nodes whose 'Highest error' is bigger than *threshold* are considered, and among all the edges stored at each node, only those edge whose error range includes *threshold* are rendered. The process is described by algorithm in figure 7.

The `render_edge(n:node,h:threshold)` routine renders edges (sides) in node n whose error range contains *h*. This is done by rendering the associated quadrilateral as two triangles.

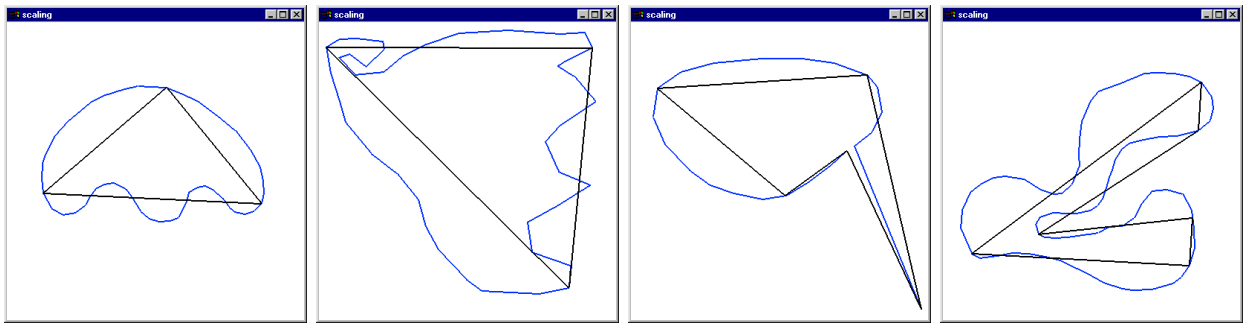
RESULTS

We have implemented the construction and rendering algorithms for MRBSP trees. Figure 8 shows different approximations of four objects for five different threshold values. Along with each subfigure rendering times are presented. Figure 9 shows the results obtained when rendering the whole object represented with a conventional BSP tree, compared to the results obtained with our MRBSP tree using our algorithm described in *Rendering* section.

This 2D serve us to check the improvement that we can expect from developing 3D MRBSP trees compared to traditional BSP trees.

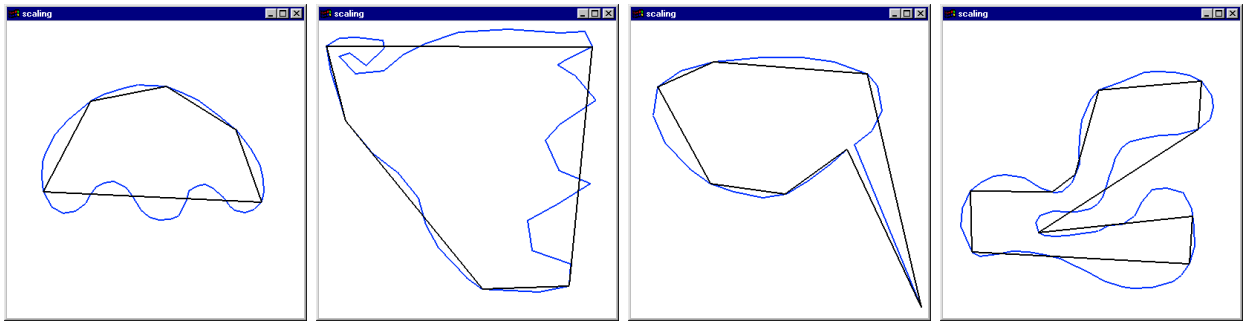
For this experiments we have used a single threshold for the whole tree (object). It would be possible to use a variable threshold for every tree branch based on distance and orientation from viewpoint. This will be done for the 3D extension.

Threshold=0.33



1,08 ms *1,12 ms* *1,34 ms* *1,33 ms*

Threshold=0.16



1,34 ms *1,38 ms* *1,45 ms* *1,54 ms*

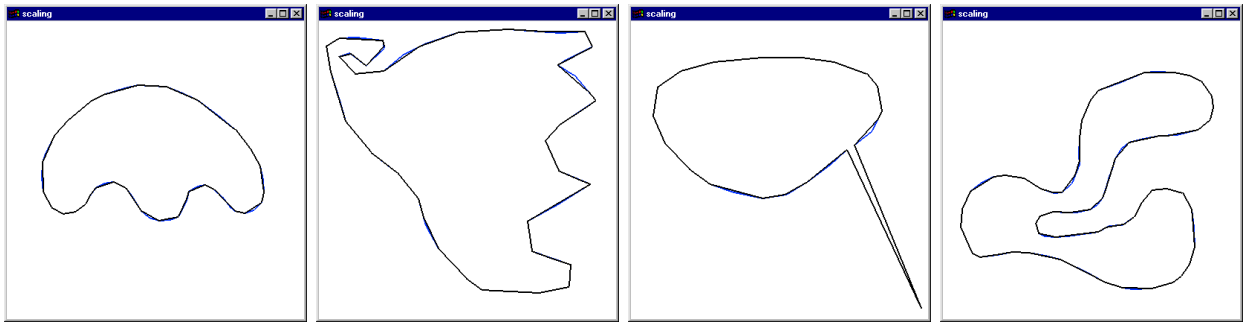
Threshold=0.08

1,86 ms *2,10 ms* *1,75 ms* *2,04 ms*

Threshold=0.04

1,79 ms *3,15 ms* *2,31 ms* *2,81 ms*

Threshold=0.01



2,40 ms *4,70s* *2,76 ms* *3,43 ms*

Figure 8. Different approximations of some 2D objects.

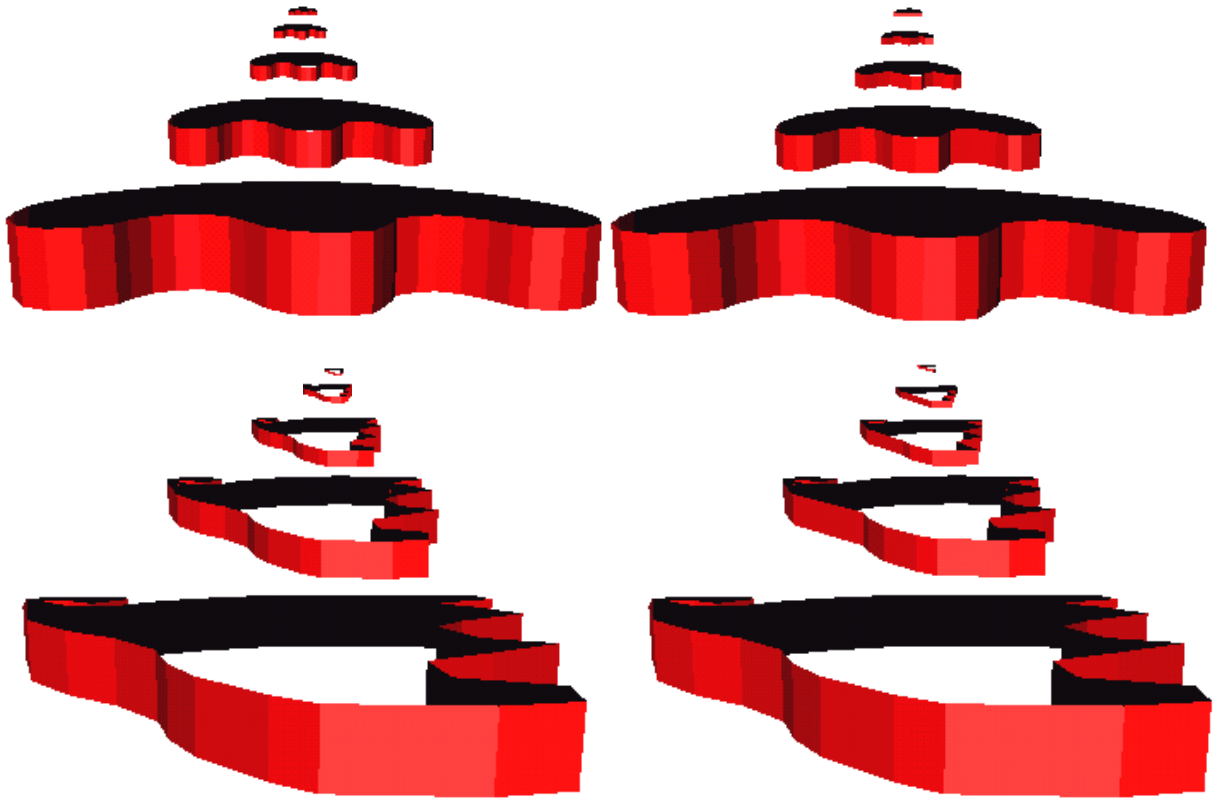


Figure 9. Left: Original object seen from different distances
Right: Different approximations of the same object seen from different distances.

CONCLUSIONS AND FUTURE WORK

This work presents preliminary results in the development of MRBSP trees for 3D space. Solving the problem in 2D is only the first step before attempting to solve the 3D case, our ultimate goal. In this case, partition planes at tree nodes placed along the edges in 2D turn into planes along a 3D face; planar regions at tree leaves turn into volumes. Finally, we expect to obtain better results by combining 2D MRBSP trees for face (polygon) representation, and 3D MRBSP trees for polyhedron representation.

For the 3D case, in order to achieve a variable resolution in our representation, it is possible to use a variable threshold, i.e., a new threshold can be computed for every tree branch, based both on distance and orientation.

Besides hidden surface removal, we would also like to apply our representation to other areas where BSP trees have been used. Specifically, we would like to use multiresolution BSP trees for speeding up space classification, raytracing, shadow computations, and, possibly, CSG operations. Some of these applications, however, require the construction of balanced BSP trees.

Finally, let us mention that our extension also improves on previous applications of BSP trees. It combines the advantages of both space partitioning and multiresolution techniques. So it is an excellent substitute for other LOD techniques, especially in applications related to real-time rendering of complex geometric models, virtual reality systems, and distributed environments.

REFERENCES

Fuchs, H., Kedem, Z., Naylor, B. 1980 On Visible Surface Determination by a Priori Tree Structures, *Computer Graphics*, Vol. 14, no 3, June, pp 124-133.

Gordon, D., Chen, S. 1991 Front-to-back display of BSP trees, *IEEE Computer Graphics and Applications*, pp 79-85, Vol. 11, no 5, September.

Heckbert, P., Garland, M. 1994 Multiresolution Modeling for Fast Rendering, *Proceedings of Graphics Interface'94*, pp 43-50.

Heckbert, P., Garland, M. 1997, Survey of Polygonal Surface Simplification Algorithms, *Course Notes, Course 25: Multiresolution Surface Modeling*, SIGGRAPH'97.

Hoppe, H. 1996 Progressive Meshes, *Proceedings of SIGGRAPH '96*.

Hoppe, H. 1997 View-Dependent Refinement of Progressive Meshes, *Proceedings of SIGGRAPH '97*.

Huerta, J., Chover, M., Quirós, R., Vivó, R., Ribelles, J. 1997, Binary Space Partitioning Trees: A Multiresolution Approach, *Proceedings Information Visualization (IV'97),* pp 148-154, August, London.

Paterson, M.S., Yao, F.F. 1990 Efficient Binary Space Partitions for Hidden Surface Removal and Solid Modeling, *Discrete Computational Geometry*, no 5, pp 485-503.

Perlin, K. 1984 A unified texture/reflectance model, *Advanced Image synthesis seminar notes,* SIGGRAPH'84.

Puppo, E., Scopigno, R. 1997 Simplification, LOD and Multiresolution, *Proceedings of EUROGRAPHICS'97*, Vol. 16, no 3.

Sakas, G., Gerth, M. 1991 Sampling and anti-aliasing of discrete 3-D volume density textures, *Proceedings of Eurographics'91*, pp 87-102.

Wiley, C.A., Fussell, D., Szygenda, S., Hudson, F. 1997 Multiresolution BSP Trees Applied to Terrain, Transparency, and General Objects,' *Proceedings Graphics Interface '97*, Kelowna, B.C, Canada.

Williams, L. Pyramidal parametrics, *Proceedings of SIGGRAPH'83*, pp 1-11.