Binary Space Partitioning Trees: A Multiresolution Approach

J. Huerta, M. Chover, R. Quirós, R. Vivó, J. Ribelles
Computer Science Department. Jaume I University.
Campus de Penyeta Roja. 12071. Castellón. Spain.
Tel: (+34) 64 345771 / Fax: (+34) 64 345848
e-mail: 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. However, with the advent of networked graphics applications there is an increasing need for multiresolution geometric representations. 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 the finest details of the representation. We present in this paper different algorithms to construct multiresolution BSP trees in 2D. Then we propose extensions of our methods to 3D space.

1. Introduction.

One of the main problems of visualization 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 different level-of-detail (LOD) representations for a given model. 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. Such representations are called multiresolution representations [1].

Their main advantage is that only the necessary geometric detail is used for model rendering. Furthermore, in a networked environment they can be transmitted much faster than the entire geometric description. However, there are still certain issues that need to be addressed regarding these representations: mesh simplification [2], level-of-detail modeling [3], selective refinement [4], and progressive rendering [5].

Certain modeling techniques, like space partitioning, allow more efficient representations than traditional techniques. The best example is Binary Space Partitioning (BSP) trees [6] 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.

Our work combines the advantages of partitioning techniques and multiresolution representations by introducing multiresolution BSP trees. Our representation allows fast spatial classification due to its BSP tree organization, and LOD control for rendering due to its multiresolution feature. Tree nodes near the root provide a coarser representation of the model, while nodes further down the hierarchy provide higher 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 graphics applications.

Section 2 introduces the problem and objectives addressed in this paper. Section 3 describes the function used to evaluate approximations and the method used to split concave polygons. Section 4 presents different construction methods for 2D multiresolution BSP trees. Section 5 compares the results obtained with those methods. Finally, section 6 presents our conclusions plus several ideas for future work.

IEEE Conference on Information Visualization (IV '97)
1093-9547/97 $10.00 © 1997 IEEE

2.BSP trees as multiresolution models

Storage of multiresolution models requires data structures that allow retrieval of LOD representation according to eye location and view orientation. Different data structures have been proposed in the literature for this purpose, like image pyramids [7], volume pyramids [8], textures and reflectance [9], polygonal models [10].

Polygonal models are by far 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. Several simplification algorithms have been proposed in the literature [10] [11] [5] but none of them includes a space partitioning representation as part of the model.

The main contribution of our paper is the use of BSP trees to store the geometry of a multiresolution representation. BSP trees allow for fast retrieval of LOD representation and fast rendering of that representation. Also, they allow for progressive transmission and rendering by traversing the tree in order. This is possible because coarser LODs are located near the root of the tree, while finer LODs are located further down the tree. The following figures clarify these issue. Figure 2 illustrates how this tree can be reconstructed in a multiresolution way.

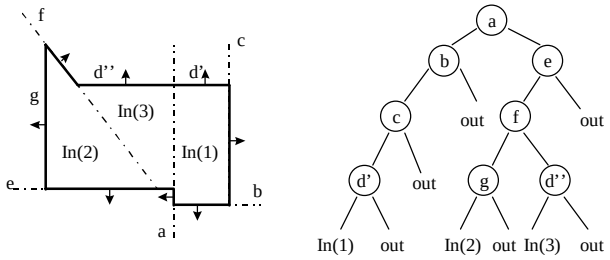


Figure 1. BSP trees for polygon representation.
Left: polygon and planes defining its boundaries.
Right: the associated BSP tree representation.

Figure 1 shows a conventional BSP tree. 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 coarser representation, while the rest of nodes add detail until the entire tree has been considered. Section 4 describes different methods to construct such trees in 2D.

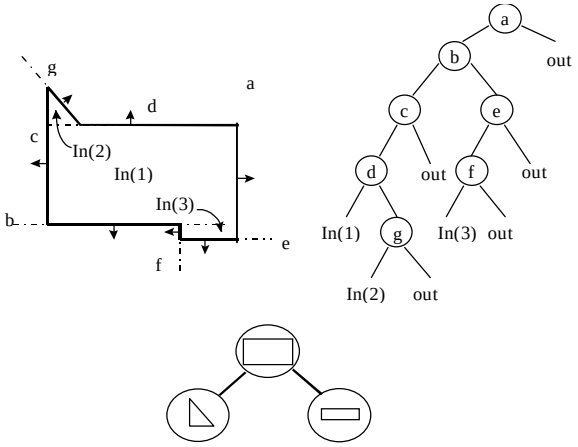


Figure 2. A multiresolution BSP tree polygon representation.
Upper left: polygon and boundary planes.
Upper right: associated BSP tree representation.
Bottom: multiresolution tree of the polygon.

3. Preliminaries

3.1 Approximation Error

Given a polygon, we want to construct its BSP tree representation by choosing partitioning planes along its edges. Thus, the final representation should contain all the edges of the polygon, plus some others that help 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 so that the amount of detail added by each choice gets smaller as the algorithm progresses. 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. One of these cases 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 later. This requires adding extra partitioning planes to the representation. Another case is to include planes that decimate edges of the original polygon.

We consider two parameters when constructing a multiresolution BSP tree representation: error incurred by the approximation, and number of cuts on the original polygon. By cuts we mean intersection points between edges and partitioning planes. Every time a new partitioning plane is added some edges may need to be

IEEE Conference on Information Visualization (IV '97)
1093-9547/97 $10.00 © 1997 IEEE

cut in half. Such cuts increase the number of edges of the polygon and thus the size of the BSP tree. Our plane selection algorithm will try both to minimize error and to avoid introducing cuts.

To evaluate this error it is necessary to develop an error function. The problem can be informally stated as follows: given a polygon *S*, represented by a set of $N \geq 2$ ordered points in the plane, and a constant *M*, $2 \leq M \leq N$, find a new set of M points that define a new polygon *R*, which most closely fits the given set.

The following notation is introduced to allow for a formal characterization of this problem:

A *sequence of points* in R^2 is a finite non-empty ordered set of coordinate pairs; i. e.,

$$S=(s_1,s_2,...,s_N)=(sx_1,sy_1),\ (sx_2,sy_2),\ldots..\ ,(sx_N,sy_N)$$

$$R=(r_1,r_2,...,r_M)=\ (rx_1,ry_1),\ (rx_2,ry_2),\ldots..\ ,(rx_M,ry_M)$$

If r_i and r_{i+1} are two consecutive vertices of *R*, then $e_i(r_i,r_{i+1})$ will denote the *edge* from r_i to r_{i+1}. Then *R* is said to be a valid approximation of *S* for this problem if $e_i(r_i,r_{i+1})$ $1 \leq i \leq M$ of *R* passes through $s \in S$. This restriction is assumed to minimize number of cuts. This will happen if:

$$\exists\, u_s,u_t \in [0.1] / \begin{cases} s_j = r_i + u_s \cdot (r_{i+1} - r_i) \\ s_k = r_i + u_t \cdot (r_{i+1} - r_i) \end{cases} \quad s_j,s_k \in S, \quad r_i,r_{i+1} \in R$$

Once *R* is defined, it is necessary to define the set of points of *S spanned* by each edge of *R* to evaluate the approximation error. Using an error measure, (i.e. square distance), we can compute the error for each edge as the sum of the errors of all the vertices *spanned* by this edge.

Given that we force edge e_i to pass through two vertices of S, let say s_i and s_j, the set of vertex $S_i:s_j,\ldots,s_k$ are *spanned* by e_i. Regarding edges e_{i-1}, e_i and e_{i+1} of *R* they approximate $S_{i-1}:s_f,...,s_g$, $S_i:s_j,\ldots,s_k$ y $S_{i+1}:s_l,...,s_m$, respectively. If $u_s \neq 0$ and $u_t \neq 1$ ($r_i,r_{i+1} \notin S$) then, two sequences $S_a:s_{g+1},...,s_{j-1}$ y $S_b:s_{k+1},...,s_{l-1}$ exist and we must find which edge spans vertices in these sequences. Thus, we define V_i as the set of vertices spanned by edge e_i, including S_i and those vertices of S_a and S_b located closer to e_i than to e_{i-1} or e_{i+1}.

Now, we can associate an error or distortion measure to each edge, as the degree of fitness of the edge to all of its spanned vertices:

$$\Delta(V_i) = \sum_{\forall v \in V_i} d(v, e_i(r_i, r_{i+1})) \quad 1 \leq i \leq M$$

where $d(v,e_i(r_i,r_{i+1}))$ is an elementary distortion, i.e. square distance , of vertex *v* with respect to the edge e_i.

So, we can measure the error of approximating polygon *S* with triangle *R* as:

$$\hat{D}(R,S) = \sum_{i=1}^{M} \Delta(V_i)$$

Several functions *d* may be used to measure the error contribution for each vertex of R. If distance (or square distance) is used, given a spike in the polygon, error is computed regardless of its width. To solve this problem it would be necessary to discretize the edges of the spike and add the distances to all points obtained in the discretization. Obviously, this may be achieved by just computing the area of the spike. Then, we use an error function that computes the area of the error incurred in the approximation as the area of the differences between the polygon *S* and the approximation triangle *R*. That error function is:

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

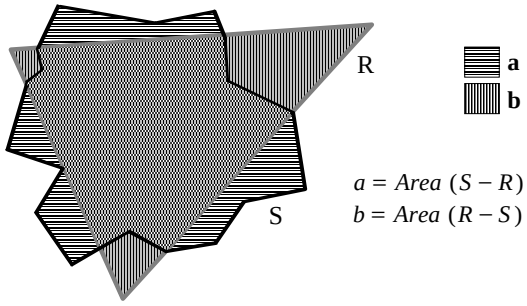


Figure 3. Computing approximation error of S with R.

3.2 Concave Polygon Subdivision

Most of the algorithms described in this paper require special treatment of highly concave polygons. Our strategy for dealing with this problem is a divide-and-conquer strategy. The idea is to split concave polygons into smaller polygons that either are convex or have less concavities. To achieve this goal we select partitioning planes that divide the polygon instead of bounding it. Such planes are selected by connecting pairs of vertices of the original polygon.

To select the pair of vertices to use in splitting process we have considered some techniques to infer the structure of a polygon. Specifically, we studied techniques based on medial axes and Voronoi skeletons [12]. However, we could not extract the necessary concavity information from this structures to appropriately subdivide a given polygon. Therefore, we decided to adopt a strategy that considers all possible pairs of vertices and determines

IEEE Conference on Information Visualization (IV '97)
1093-9547/97 $10.00 © 1997 IEEE

which one defines the best partitioning plane. In order to reduce the amount of work necessary to evaluate all pairs, we only consider certain candidate pairs. Candidate pairs must satisfy all of the following conditions:

1. they must not be endpoints of the same edge,
2. the partitioning line they define must divide the polygon into two subregions, and
3. at least one of the vertices must not belong to the polygon's convex hull.

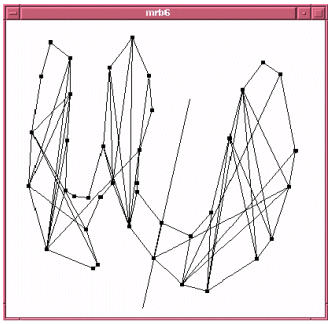


Figure 4. Subdividing Concave Polygons

It is possible that no vertex pair of the polygon satisfies all of the above conditions. In that case we add new vertices to the polygon by inserting all the midpoints of its edges in its vertex list. Then we run the above algorithm again, and recursively repeat the process until some candidate pair is found. If we find more than one candidate pair, then we select the best one using the difference area between the polygon and its convex hull.

In some cases, when a concave polygon needs to be subdivided, it is possible that the sequences of approximations associated to each subregion do not exactly match with each other. In such cases we solve the problem by imposing certain constraints on the way the approximations are constructed to assure that the sequence of approximation of the parts fit together properly.

4. CONSTRUCTION METHODS

This section presents four methods developed to construct multiresolution BSP trees. All methods start by computing the initial approximation as the triangle that minimizes the error function defined in section 3.1. Edges of this triangle will be inserted on top of the tree as partitioning planes. If those planes cut some edge of the original polygon, then we add new vertices to its representation. Next we classify all the vertices with respect to the planes already in the tree, obtaining several sequences of vertices that are located in the same region. New polygons are constructed with each one of this sequences and then, the algorithm is applied recursively to each of these new polygons to generate subsequent planes till the original polygon is fully represented.

To force subsequent planes fit previous ones properly, we constrain triangle approximation for each one of the new polygons to pass through first and last vertex of the sequence. This constrain is carried out by applying some restrictions that simplify the process. Since the first and last vertices of the sequence are located over planes already placed in the tree, only two new planes will be inserted in each recursion. This is described below for each method.

4.1. Three vertices approximation

Computing the initial approximation. In this method we construct an initial approximation by choosing three vertices of the polygon. All possible triangles that share three vertices with the original polygon are evaluated, selecting the triangle that minimizes our error function. For convex polygons we obtain the largest possible triangle enclosed in the polygon (see figure 5a).

Computing subsequent planes: Given that we force triangle approximation for each polygon to pass through first and last vertex of the polygon, these must be vertices of the triangle. Then, only one vertex is left to be chosen. We add two new planes in the tree along the line that join this vertex and the first and last ones of the sequence.

Analysis. This method produces acceptable results in terms of error and temporal cost. For convex polygons no cuts are produced. In this case, what we get is a constrained triangulation [13] such that contains the largest possible triangle enclosed in the polygon plus a set of smaller triangles around it, that provide successive approximations to the original polygon boundary (see figure 5a). Concave polygons (figure 5b) give rise to a great number of cuts that enlarge size of the resulting tree. Therefore, this method produces also the highest number of planes. Complexity of this algorithm is $O(n^3)$.

4.2. Three edges approximation

Computing the initial approximation. In this method we construct an initial approximation by choosing three edges of the original polygon. The process is similar to three vertices approximation, but evaluating all the triangles that can be obtained by extending any three edges of the original polygon. For convex polygons we obtain the smallest possible triangle that surrounds the entire original polygon (see figure 5b).

Computing subsequent planes: We restrict our search to two of the edges obtained by joining all possible pairs of vertices of the sequence. The error function is evaluated for this pairs and all the planes already in the tree and the pair with minimum error is selected. Then this pair is inserted in the tree as two new planes.

Analysis. This method is specially suited for convex polygons giving certainly good results. On the other hand, for concave polygons results are not so good. Its main advantage is that planes are always placed through the edges of the original polygon and this keep size of the tree to a minimum.

4.3 Six vertices approximation

Computing the initial approximation. In this method we only consider triangles that intersect the original polygon at six of its vertices. First, new edges are constructed joining every pair of vertices in the original polygon. Those edges are then extended until they intersect forming triangles. The error function is then evaluated for each possible triangle, and the triangle with the smallest error is selected as the initial approximation.

Computing subsequent planes: Again, the restriction used is that the edge joining the first and last vertices of the sequence must be in the triangle. Then we construct new edges joining the rest of vertices and, among them, we choose those two that form the triangle (together with the already mentioned) with minimum error. After that, this two new planes will be inserted.

Analysis. This method produces substantially better results in terms of error minimization at the expense of added time complexity: $O(n^6)$. Also there are some mismatches between successive triangles. The number of cuts is similar to the other two strategies (figures 5e, 6e).

4.4 Scaling approximation

The scaling method is based on the idea that objects further from the viewer look smaller than objects closer to the viewer. This implies that finer detail of a given object disappears as it moves away from the viewer. We can thus use this characteristic to compute different LOD representations of an object.

Computing the initial approximation. 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, then we substitute them by a new single edge. This procedure is repeated for decreasing scale factors until there are only three edges left. 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 one can substitute the two old edges.

Computing subsequent planes: Likewise three vertices approximation method, we use a restriction that force the edge joining the first and last vertices of the sequence to be in the triangle, thus its colinearity is not tested. So we check colinearity of the rest of edges removing one of them at a time till there are only two left. Then, these ones are inserted in the tree.

Analysis. Both error and number of cuts obtained by this algorithm are comparable to those produced by the above methods (see figures 5d and 6d). However, its time complexity is much better: $O(n^2)$. For that reason we believe that this method is particularly good at constructing BSP trees for 2D polygons.

Another advantage of this method is that we can associate a distance to each of the nodes of the BSP tree. The partitioning plane associated to that node will then be used for rendering when the distance from the viewer to the object is smaller than the node's distance. Such distances can be easily computed from the scale factors used in the approximation algorithm. The resulting structure is a multiresolution BSP tree whose LODs are already parameterized according to the distance from the object to the viewer.

5. RESULTS

We have implemented the algorithms described in the previous section. Figures 5 and 6 show the results obtained by applying each of the methods to two test polygons, respectively. First polygon is convex while second one is concave. Figure 7 contains a comparison of the performance of the different methods. We can draw the following conclusions from the graphs in figure 7:

- **Error of the initial approximation**. The best method is the six-vertex method, but we discard it for cost reasons. Three-vertices, three edges and scaling methods present an acceptable error ratio.

Time complexity. Results presented in this work are produced by greedy algorithms with no optimization. It is possible to reduce the order of their cost at least by one using some techniques like incremental error computation presented in [14]. This has not been done yet, as this work is just a preliminary study. The best results are obtained by the scaling, with $O(n^2)$ cost.

Three vertex and three edges approximation methods have also an acceptable cost, $O(n^3)$. Even though this is not an key factor (recall that BSP trees are constructed during a preprocessing stage), the six vertices method require too much time to compute, especially, when method be extended to 3D.

- **Number of cuts**. The best results are obtained with the scaling approximation method. The three-vertex method produce acceptable results, too.

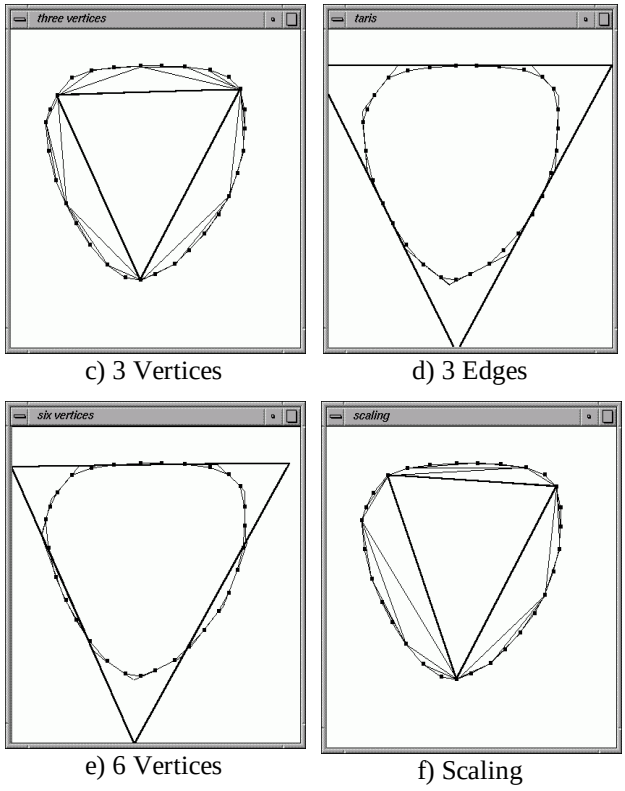
- **Number of tree planes**. The best approximation is the three-edge approximation because partitioning planes always coincide with edges of the original polygon. Results of the rest of methods are acceptable.

6. CONCLUSIONS. FUTURE WORK.

From the above results we can conclude that the three edges approximation is the best one for convex polygons. However, the scaling method is the most general of all the methods presented in this paper. Additionally, it produces good intermediate approximations, and preserves the most significant vertices of the original polygon. Furthermore, we can stop the scaling process when a minimum number of edges is reached, like four or five, instead of three. An additional feature is the distance parameter that we can store with each node in the BSP tree. Finally, the most important reason to adopt the scaling method is the fact that it produces far better visual results than the rest of approximation methods. Hence, our next step is to try to extend it to 3D. We expect such extension to be easier for the scaling method than for the rest of methods.

As immediate future work we are extending the method to 3D. In this case, partition planes at tree nodes placed along edges in 2D turn into planes along a face; planar regions at tree leafs turn into volumes. Finally, we can obtain better results by combining 2D BSP trees for polygon representation, and 3D BSP trees for polyhedron representation.

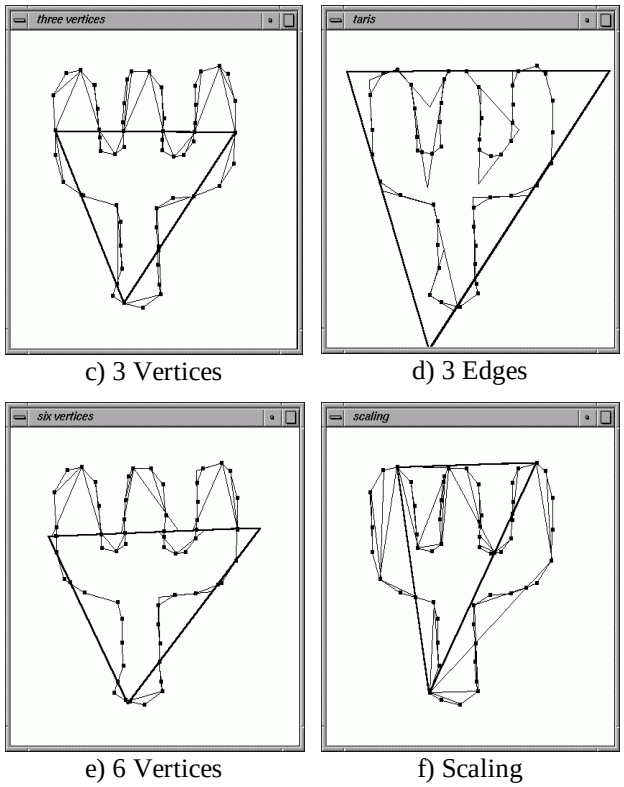
We expect to extend to 3D all our approaches but the six-vertex method due to the problems they pose in 2D. We will analyze possibilities of the three-vertex and the three-edge methods, and we will certainly develop extensions for the scaling method. We would also like to apply our representation to 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.



c) 3 Vertices d) 3 Edges

e) 6 Vertices f) Scaling

Figure 5. Construction methods for convex polygons.



c) 3 Vertices d) 3 Edges

e) 6 Vertices f) Scaling

Figure 6. Construction methods for concave polygons

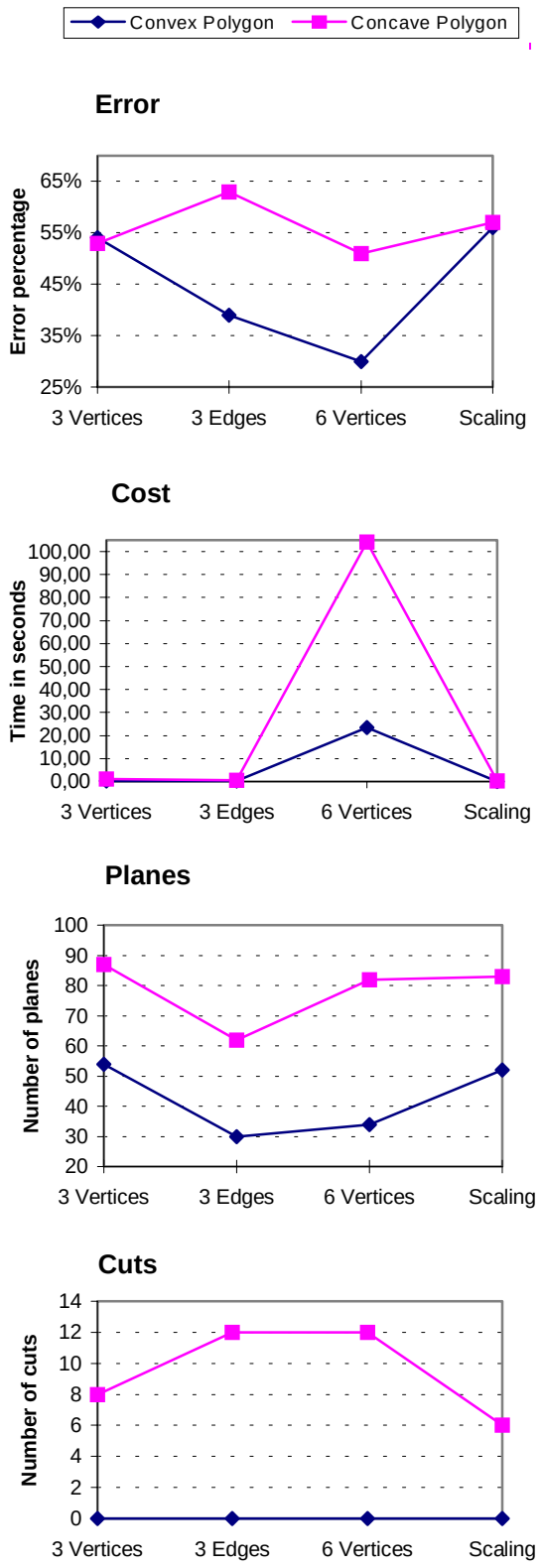


Figure 7. A comparison between the different BSP tree construction methods.

Also, some work is being done to improve the error function. For example, consider a polygon with a long narrow spike. Even though the spike may have a small area, its presence or absence from the image may be important. To solve this problem we are considering the use of some techniques as the method of moments [15].

Finally, let us mention that our extension greatly 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, complex geometric models, virtual reality systems, and distributed environments (VRML).

REFERENCES

1 Paul S. Heckbert, Michael Garland, "*Multiresolution Modeling for Fast Rendering*", Proceedings of Graphics Interface '94. pages 43-50.

2 Hugues Hoppe et al, "*Mesh Optimization*", Computer Graphics, Proceedings of SIGGRAPH '93, pages 19-26.

3 P. Astheimer, M.L. Peche. *"Level-of-Detail Generation and its Applications in Virtual Reality"* Proceedings of VRST'94, August 1994, pages 299-312.

4 M. Pérez et al, *"Gestión de niveles de detalle y morphing para representación de terrenos en tiempo real*", CEIG'95, Palma de Mallorca, June 1995, pages 195-208.

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

5 Hugues Hoppe, "*Progressive Meshes*", Computer Graphics, Proceedings of SIGGRAPH '96, 1996.

9 Ken Perlin. "*A unified texture/reflectance model*", SIGGRAPH'84, Advanced Image synthesis seminar notes, July 1984.

7 Lance Williams. "*Pyramidal parametrics*", Proceedings of SIGGRAPH'83, July 1983, pages 1-11.

8 Georgios Sakas and Mathias Gerth. "*Sampling and anti-aliasing of discrete 3-D volume density textures*", Proceedings of Eurographics '91, pages 87-102.

10 William J. Schoroeder, Jonathan A. Zarge y William E. Lorensen. "*Decimation of triangle meshes*", Proceedings of. SIGGRAPH'92, Julio 1992, pages 65-70.

11 Greg Turk. "*Re-Tiling polygonal surfaces"*, Computer Graphics (SIGGRAPH'92), vol. 26, nº 2, pages 55-64.

12 Franco P. Preparata, Michael Ian Shamos. "*Computational Geometry, An introduction"*, Springer Verlag, 1985, pages 205-225

13 Chew, L. Paul. "*Constrained Delaunay triangulations*", Algorithmica 4 (1989), N 1, pages 97-108.

14 J. C. Pérez, E. Vidal, "*Optimum polygonal approximation of digitized curves*", Pattern Recognition Letters 15, pages 743-750, 1994.

15 Cho Huak Teh, Roland T. Chin, "*On image analysis by the methods of moments*", IEEE Transactions on pattern analysis and machine intelligence, vol. 10, N 4, 1983.