Efficient Object BSP Trees

Navendu Jain, Sorav Bansal, Sanjiv Kapoor
Computer Science and Engineering Department,
Block VI, Indian Institute of Technology,
Hauz Khas, New Delhi 110016, India.
Email: skapoor@cse.iitd.ernet.in

Abstract

In this paper we investigate Object Oriented Binary Space Partitioning. The Binary Space Partition Tree (BSP-Trees) is a widely used and effective data structure for solid modeling and hidden surface removal. We present algorithms for efficiently constructing Object BSP Trees(OBSP) in 2-Dimensions. The term object used arises since the construction of our tree utilizes the property of hierarchical ellipsoid covers. Besides being object-oriented, the major advantage of the OBSP trees is the reduction in the fragmentation of the scene. This paper formulates algorithms and data-structures for representing polyhedral objects and operations such as composition, deletion and motion of objects based on Object Hierarchy using the modified version of the traditional BSP tree. We improve on the design presented in [11]. Implementation details are presented which show increased efficiency resulting through our approach.

Keywords : BSP Trees, Object Hierarchy, Fragments, Ellipsoidal cover, Separating Plane.

1 Introduction

Solid Modeling and Hidden Surface Removal are fundamental problems in both Computer Graphics and Computational Geometry. Recent years have seen a rapid increase in Graphics applications. Various algorithms are available which can efficiently do Hidden Surface Removal [5], [9] for a static scene, mostly based on Open GL. However, when changes to the scene are done dynamically, maintenance of the scene becomes inefficient.

A data structure that is often used for scene representation is the Binary Space Partition Tree [5], [6], [7], [8]. For dynamic scenes, the basic problem of Hidden Surface Removal poses even a greater challenge than ever before. Faster animation is the goal for many applications, most especially games. The inherent classical BSP(Binary Space Partitioning tree) proves to be inefficient both in terms of deletions and insertions of Objects.

In this paper we propose modifications to improve the BSP. We restrict our attention to 2-D scenes.

The Dynamic Hidden Surface Removal problem is: Given a set of Polyhedral Objects $P_1, P_2 \ldots P_n$ in 2-D and a viewing point v, determine and maintain visible surfaces under changes to the scene allowing for insertions and deletions dynamically. In the original implementation of the standard separating plane BSP tree, the separating planes are identified from the planes corresponding to the polygons comprising the scene. This data structure enables us to develop an algorithm for determining the back to front ordering of the polygons by performing an in-order traversal of the tree dependent upon the viewpoint. Though good for static environments, such a structure loses its advantages in a dynamic environment. A dynamic environment involves simulation of motion which is essentially deletions and reinsertions. Any deletion may result in a number of re-computations of subtrees of the BSP [2]. Hence, this will be a very expensive operation.

In a previous approach [10], the polygons are maintained in a space partitioning structure called the Separating Plane Partition Tree which is a variation of the traditional BSP tree. In this variation, separating planes are the fundamental objects which are used to partition the space. The polygons may be fragmented but each of the fragments is stored at the leaves. This allows for easy access of polygon fragments for efficient insertion and constant time deletion per fragment. Efficient hidden surface removal of the polygons is then achieved by painting the fragments in an order obtained from the space partitioning tree.

The structure described in [10] allows us to move from the domain of polygons to the domain of objects i.e. polyhedra. Objects are defined and constructed recursively as a polygon or a merge of two objects. Such an approach was suggested in [11]. Objects are represented as Binary Space Partition trees. These objects can be defined, added and moved independently.

The BSP tree identifying the object has all the constituent polygons at the leaf nodes. We use the variant BSP described above [11]. The Binary Space Partition tree corresponding to the entire space is obtained by merging the objects or in other words merging the corresponding BSP trees.

In order to compute the separating planes(lines in 2D) efficiently we suggest the use of the ellipse hierarchy which enclose the objects. An object is composed from sub-objects. Consequently the ellipse enclosing the object encloses the ellipses corresponding to the constituent objects. Thus a representation of the object is as a union of ellipses at any level of the object hierarchy. In this paper we exploit the hierarchy of ellipses to construct the separating plane(line) and reduce fragmentation. If a separating line between two objects is not determined from the two union of ellipses at some level i then determination of a separating line is attempted at the next level $i+1$. On failure, subsequent levels are considered upto some predefined level number based on efficiency tradeoffs. The procedure starts at level 1. We show experimentally that this reduces fragmentation. The important thing to note here is that throughout we deal with an object as the basic unit of operation. From here on, we will describe our modified BSP as the Object Ellipse BSP Tree.

We note that the concept of approximating objects by a union of spheres has also been used in [1] and collision detection[3]. Naturally, in our approach, we expect the use of ellipsoids to lead to better approximations of the shape of polyhedron and increased efficiency.

The paper is organized as follows. Section 2 presents an overview of the standard BSP tree. Section 3 describes our algorithm and underlying data structure. Section 4 presents the implementation details and results. Section 5 describes the conclusions of our work.

2 BSP Trees

A Binary Space Partitioning (BSP) tree is a standard binary tree used to sort and search for polytopes in n-dimensional space. The tree taken as a whole represents the entire space, and each node in the tree represents a convex subspace. Each node stores a **hyperplane** which divides the space it represents into two halves, and has as children two nodes which represent each half. In addition, each node may store one or more polytopes.

The BSP tree construction is a process which takes a subspace and partitions it by any hyperplane that intersects the interior of that subspace. The result is two new subspaces that can be further partitioned by recursive application of the method. A **hyperplane** in n-dimensional space is a $n-1$ dimensional object which can be used to divide the space into two half-spaces. For example, in three dimensional space, the **hyperplane** is a plane. In two dimensional space, a line is used.

Construction of BSP Trees

Given a set of polygons in three dimensional space, we want to build a BSP tree which contains all of the polygons.

The algorithm to build a BSP tree is very simple:

1. Select a partition plane.
2. Partition the set of polygons with the plane.
3. Recurse with each of the two new sets.

Choosing the partition plane

The choice of partition plane depends on how the tree will be used, and what sort of efficiency criteria we have for the construction. For some purposes, it is appropriate to choose the partition plane from the input set of polygons. Other applications may benefit from axis aligned orthogonal partitions. Choosing planes which separate two objects can also be useful. It is also desirable to have a balanced tree, where each child contains roughly the same number of polygons. However, there is some cost in achieving this. If a polygon happens to span the partition plane, it will be split into two or more pieces. A poor choice of the partition plane can result in many such splits, and a marked increase in the number of polygons.

Partitioning polygons

Partitioning a set of polygons with a plane is done by classifying each member of the set with respect to the plane. If a polygon lies entirely on one side or the other of the plane, then it is not modified, and is added to the partition set for the side that it is on. If a polygon spans the plane, it is split into two pieces and the resulting parts are added to the sets associated with either side as appropriate.

Partitioning a polygon with a plane is a matter of determining which side of the plane the polygon is on. This is referred to as a front/back test, and is performed by testing each point in the polygon against the plane. If all of the points lie to one side of the plane, then the entire polygon is on that side and does not need to be split. If some points lie on both sides of the plane, then the polygon is split into two or more pieces. The basic algorithm is to consider all the edges of the polygon and find those for which one vertex is on each side of the partition plane. The intersection

points of these edges and the plane are computed, and those points are used as new vertices for the resulting pieces.

Hidden Surface Removal

Probably the most common application of BSP trees is hidden surface removal in three dimensions. BSP trees provide an elegant, efficient method for sorting polygons via a depth first tree walk. This fact can be exploited in a back to front **painter's algorithm** approach to the visible surface problem, or a front to back **scan-line approach**.

BSP trees are well suited to interactive display of static(not moving) geometry because the tree can be constructed as a pre-process. Then the display from any arbitrary viewpoint can be done in linear time. Adding dynamic(moving) objects to the scene efficiently is a significant contribution of our work.

Minimizing splitting

An obvious problem with BSP trees is that polygons get split during the construction phase, which results in a larger number of polygons. Larger number of polygons translate into larger storage requirements and longer tree traversal times. This is undesirable in all applications of BSP trees, so schemes for minimizing splitting become extremely important.

3 Object Ellipse BSP Tree

In this section we describe algorithms to efficiently construct and maintain Object Ellipse BSP Trees. An Object is defined to be a polygon or recursively a composition of two objects. The main emphasis has been to focus on the notion of Objects so that individual motions, compositions and deletions of a set of polyhedra can be done effectively. Also when a tree like structure is maintained, the effort has been to reduce the number the splits i.e. minimum number of fragments of the polygons inserted in the tree.

3.1 Data Structure for Object Ellipse BSP Tree

The description of the data structure used to store the polygons in our algorithm is :-

- It is a binary tree which partitions the entire 2D space into regions. We denote this tree as **B**.
- All the nodes of **B** either contain lines or polygons.
- Only leaf nodes of **B** contain polygons.
- All other nodes of **B** contain lines which partition the 2D space.
- All non-leaf nodes have an ellipse cover of the underlying object ellipse tree.
- All non-leaf nodes are root nodes of the Object Ellipse trees rooted at them. Each non-leaf node has an ellipse cover found by merging the ellipse covers of its two children.
- Any node **T** has a particular region, say **R** in space associated with it.
- If **T** contains a line, then this line again divides **R** into two regions, with which the left and the right child of **T** are associated.
- If **T** contains a polygon then **T** should be a leaf node and the polygon contained in **T** should lie completely within the region associated with **T** and no other polygon should lie in this region.
- The root of **B** node is associated with the entire 2-D space.

Furthermore, we let $H_i(O)$ (the representation of the object O at level i in the object hierarchy) denote the union of all ellipses at level i. At the top level is the complete object O with a single ellipse cover.

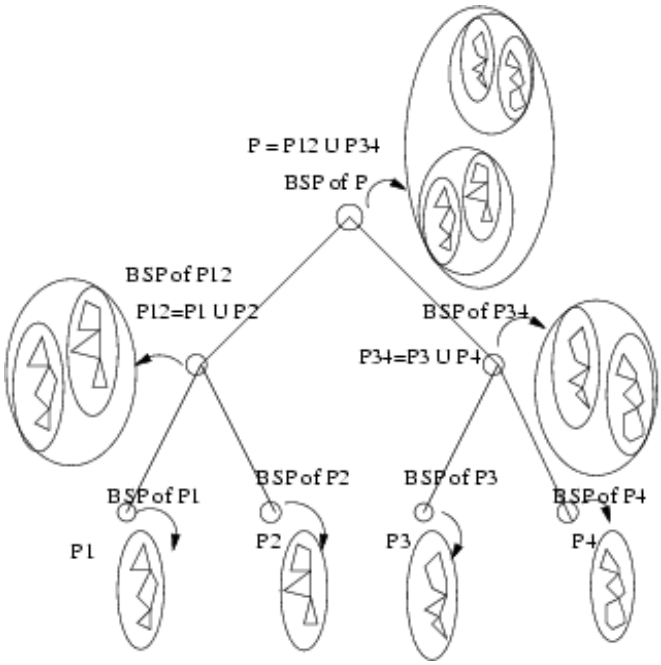

Figure 1: Object Ellipse Tree

3.2 Construction of the Object Ellipse Tree

We will describe the construction of our Binary Space Partitioning tree in two parts. The first one

concentrates on finding the ellipse covers of the objects and merging them so as to construct the hierarchical tree. The second part describes how to compute the separating line between two objects O_1 and O_2.

3.2.1 Merging algorithm for objects

Each object has its own OBSP tree. When two objects are to be merged, then a new tree representing the combined object needs to be constructed. In our strategy the efficiency of tree construction is dependent on the ability to determine a separating line between the two objects. A separating line **Pl** is defined, which divides the space into two semi-spaces such that each object lies on either side of the line.

In this section, we describe the algorithm for merging two BSP trees $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$, representing O_1 and O_2, to generate a new BSP tree $\boldsymbol{B}$ such that the root node of **B** contains the separating line of the objects O_1 and O_2 rooted at $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ respectively as well as the ellipse cover found by merging the ellipse covers of O_1 and O_2. We discuss here how to merge the two objects by combining their ellipse covers.

Let us denote the ellipse covers of O_1 and O_2 by E_1 and E_2 respectively and the ellipse cover of E_1 and E_2 to be found by E.

Step 0) Take the center C of E to be the midpoint of the line segment joining the centers of E_1 and E_2.

Step 1) Find out the maximal distant point from center of E to both E_1, E_2. This is achieved by maximizing the distance from C to a point P with the constraint that P lies on one of the ellipses. We do it separately for E_1 and E_2 and take the maximum of the two. We describe the case of E_1. The case of E_2 will be exactly similar. Given E_1 in an arbitrary orientation, we apply translation and rotation transformations to align it with the axes and apply the same transformation on C to get C' in this new space. The equations become :

$$Maximize \quad (C'_x - x)^2 + (C'_y - y)^2$$

subject to the constraint

$$ax^2 + by^2 = 1$$

This is solved by Langrange's Method. Suppose the inverse transformation applied to P maps to P', then the line CP' will define the semi-major axis of the ellipse E.

Step 2(a)) Since we want our tree construction to be efficient, we would ideally like to compute the minimum area spanning ellipse E. We determine the minor axis of E keeping in mind this goal. We give two approaches to determine the minor axis of the ellipse. We first present the simpler approach, which we implemented. The other approach is described in the Step 2(b).
Here without loss of generality, assume that the maximal distant point from C to the two ellipses lies on E_1 denoted by P_{E_1}. Denote the farthest point from C to E_2 by P_{E_2} and the distance of P_{E_1} from C to be M_c. Take the projection of the vector $\mathbf{CP_{E_2}}$ on the unit vector perpendicular to the major axis $\mathbf{CP_{E_1}}$. Taking this projection as the lower bound and M_c as the upper bound, we do a binary search to determine the minor axis. For a value of the minor axis within this range, the binary search checks for intersection of E with both E_1 and E_2 and modify the range until the interval becomes very small. Determining the intersection of two ellipses is described at the end of this Procedure. Go to Step 3.

Step 2(b)) In this approach, we have two subcases. Without loss of generality, assume that the farthest point found in previous step was on E_1. Then in the first case, ellipse E_2 touches E and is contained in E and E includes E_1 also. The other case is similar being that E_1 itself guides the cover so that E_2 lies wholly inside E and E_1 touches E.

We consider the former sub-case described above. First, we do a transformation such that E becomes aligned with the coordinate axes and E_2 becomes a circle. At the point of contact of E_2 and E, both $\frac{\delta f}{\delta x}$ and $\frac{\delta f}{\delta y}$ of E_2 should be related by a constant factor to those of E. Also, the contact point should lie on both the circle E_2 as well as the ellipse E so that the equations become :
for $E \quad f_1 : ax^2 + by^2 - 1$
for $E_2 \quad f_2 : (x - x_0)^2 + (y - y_0)^2 - R$
The conditions give the following equations :

1. $2ax = 2\lambda(x - x_0)$
2. $2by = 2\lambda(y - y_0)$
3. $ax^2 + by^2 = 1$
4. $(x - x_0)^2 + (y - y_0)^2 = R$

where λ is a constant to be determined. These equations can be solved numerically to calculate x, y, b, λ. Then from the calculated values of b,we take the maximum one to be the length of the

minor axis. Call this M_1.
Follow exactly the same procedure for E_1 and E. Denote the calculated value as M_2. Take the maximum of M_1 and M_2 as the length of minor axis. Go to Step 3.

Step 3)We have determined the ellipse cover. Exit.

To determine the intersection between two ellipses E_1, E_2 in arbitrary orientation, we transform one of the ellipses, say E_1, into a circle by compression of the axes. This is achieved by first bringing it to the origin and rotating the axis for aligning. The other ellipse E_2 is altered using the same transformation matrix. Now, we bring E_2 to the origin and align it with the coordinate axis. Consequently both of them are in the form of :

$$ax^2 + by^2 = 1$$
$$(x - x_0)^2 + (y - y_0)^2 = R$$

which can be solved in constant time (in parametrized form) to calculate the intersection points so as to determine whether E_1 & E_2 intersect or not.

3.2.2 The Separating Line Algorithm

We attempt to find the separating line between the ellipse covers E_1 and E_2 of objects O_1 and O_2 respectively. If a separating line is not determined from the ellipses of $H_i(O_1)$ & $H_i(O_2)$ (to compose object O), the separating line is constructed from $H_{i+1}(O_1)$ and $H_{i+1}(O_2)$. We present the algorithms for determining the separating line between two ellipse covers and m, n ellipse covers subsequently.

Separating line between two ellipses

We compute the midpoint P of the line segment joining the centers of the two ellipse covers E_1 and E_2. From P, we construct tangents to the two ellipses. So in all, we have a maximum of four tangents. We search for the separating line by moving the point P left or right by a binary search procedure which reduces the range of the search by moving the point away from the ellipse to which we can't find a tangent. We take the mid-point of centers of the ellipses as the starting point of our binary search and initialize lower and upper bounds for the binary search as the centers of left and right ellipse respectively.

Claim: If the ellipses can be separated, then there exists a separating line which is (at-least) one of the four tangents.

Proof: Suppose a separating line exists. Simply rotate the separating line at that point of intersection of the separating line with the line joining the centers, until it touches an object without intersecting the other one.

In case we are not able to find the separating line between the two ellipses which implies that the two ellipses intersect, we use the M-N technique described next. Here, instead of finding the separating line between $H_1(O_1)$ and $H_1(O_2)$ themselves, we go down the hierarchy to find the separating line between $H_i(O_1)$ and $H_i(O_2)$ where
$i = 2, \ldots, \min(height(H_1(O_1)), height(H_1(O_2)))$

The Object Hierarchy is exploited to decompose an ellipse cover into smaller ellipse covers of its children to find the Separating Line when the ellipse covers of the two Objects O_1, O_2 intersect. The problem essentially reduces to finding a Separating Line between m ellipses on one side and n ellipses on the other. These m, n ellipses are the smaller objects in hierarchy.

Separating Line between groups of ellipses: The coordinates of the centers of two objects are obtained as the mean of the coordinates of the individual ellipses comprising each object. We denote these centers by C_1, C_2.

Hypothesis: Suppose a separating line exists. Then there exists a point on the line joining the centers C_1, C_2 such that the tangent from that point to one of the objects is a separating line.

We call this the M-N technique to find the separating line between m ellipses constituting an object and n ellipses constituting another object. Join the centers of the two objects. We search for the point of intersection of the separating line S with the line L joining the centers by a binary search procedure. We need a decision procedure to figure out which way to move on the line L to find the required point P to determine the separating line. We construct tangents to ellipses of both the objects from this point P. Following are the possible cases that arise:

- CASE 1: (Figure 2) If the separating line is found out then Exit.
- CASE 2: No tangent exists from the point P to the ellipses of an object say O_1 (without loss of generality). This implies that the point P lies inside the object O_1. So, we move away from O_1 to a point bisecting the line segment PC_2.
- CASE 3: (Figure 3) The tangent to an object say O_1 from P intersects the object itself. In this case

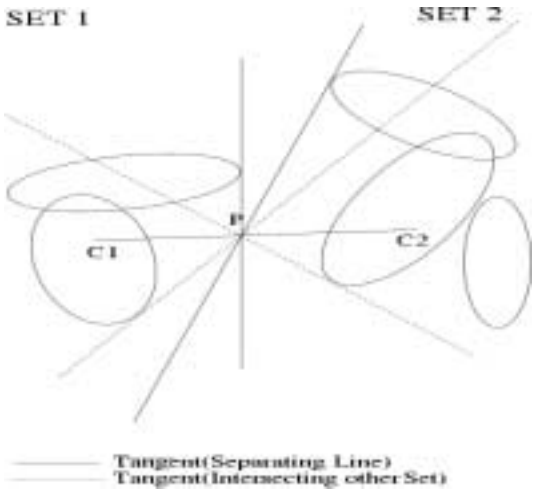

Figure 2: Separating Line

also, we move away from O_1 as in Case 2.

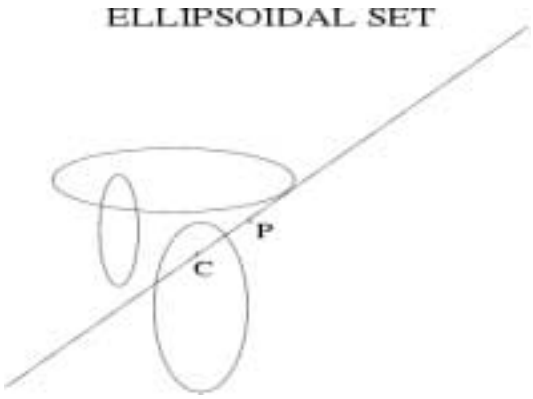

Figure 3: Self Intersecting Tangent

- CASE 4:(Figure 4) One of the four tangents is obstructed by ellipses of an object, say a tangent from P to O_2 intersects O_1 before the point of tangency. In this case again, we move away from O_1.

For each point on the line L, the number of intersection-tests between the separating line and the ellipses as well as the tangent construction operations to ellipses are $O(m+n)$. The required point on L is found in $log(|L|)$ steps where $|L|$ is the length of the line segment L. Thus, the time complexity of the above procedure is $O((m+n)log(|L|))$.

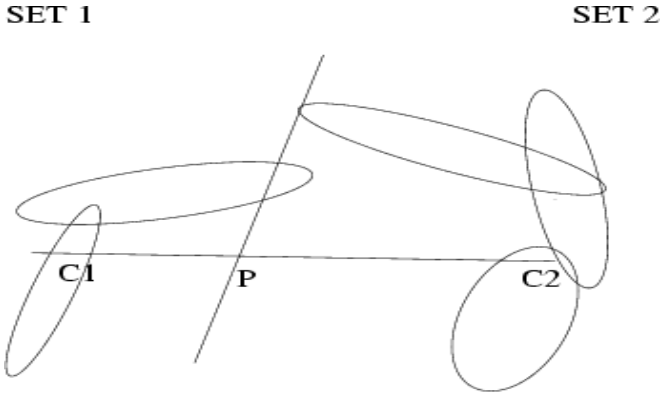

Figure 4: Intersection with Object before the point of tangency

3.3 Insertion in the Tree

For the base case of the polygon, find its ellipse cover and make an Object Ellipse Tree with the polygon at the leaf node and the ellipse cover at the parent. When we merge two objects(OBSP), O_1 and O_2 to obtain the OBSP of O, we search for the separating line P_{sep} upto a predefined depth using the Separating Line algorithm. If we are able to find the separating line P_{sep}, the OBSP tree of O is constructed with P_{sep} at the root node and O_1, O_2 as its two children. On failure, we insert one of the objects, say O_2 in the OBSP tree of O_1 taking the separating line P_l of O_1 as the separating line of O at the root node. In case P_l intersects O_2, we partition O_2 by P_l, inserting the polygon fragments of O_2 into its children recursively. The ellipse covers of subtrees of O_1 affected by insertion are recomputed to obtain OBSP tree of O. Then using the above Merging and Separating Line Algorithms, we can build the tree hierarchy recursively given a method for combining polygons into objects.

3.4 Deletion of an Object

When an object is deleted, it is also deleted from its parent object (if any), and this modification is propagated up. Deletion of an object from its parent is a linear time operation if we maintain with each object a pointer to the first polygon in the tree and all the polygons of the object are linked sequentially within the tree. Each polygon in turn has a link list of its fragments in the OBSP tree.

3.5 Deletion of a Polygon :

Since a polygon aimed for deletion may be fragmented while insertion, we maintain a thread of fragments of the same polygon within the BSP.

To delete a polygon

- move to the first fragment of the polygon
- delete the node containing it
- replace the parent of this node by the node containing the sibling of this node
- repeat this for all fragments of the polygon

Note that since the parent of this node contains a separating line between it and its sibling we do not require the line any more after the deletion of this node. Hence we just leave the sibling. This leaves the tree "dirty" which can be cleaned up at a subsequent stage.

The time for deleting a fragment is $O(1)$ and for deleting an entire polygon the time taken is $O(f)$, where f is the number of fragments of the polygon.

3.6 Simulating motion :

The algorithm which uses this BSP tree to simulate motion of a polygon would do so by deleting the polygon from the tree and then inserting a polygon corresponding to the new position of the moved polygon. If the motion is by a small distance, the separating lines in the tree would almost remain the same as computed initially, except for the ones which are the immediate parents of the fragments of the moving polygon.

In the original BSP, an internal node contains a polygon in the scene with its plane as the separating plane. Such an approach would imply that at each deletion, we regenerate the entire subtree lying below the separating line formed by the moving polygon, and do so for all fragments of the polygon. In our BSP, since all polygons lie at the leaf nodes, deleting a fragment becomes a constant time operation, and is thus much more efficient.

When an object moves, the complete BSP tree is updated though the structure remains almost the same. Though, very much optimized, the motion of an object could result in redundant nodes containing separating lines in the parent object, after a number of iterations. This could be checked by periodic reconstruction of the complete structure i.e. after a fixed number of iterations.

4 Results

The algorithm presented was implemented and tested for the fragmentation it induced on the polygons in the scene.

Random data was generated as clusters of polygons. The polygons were combined in a hierarchy to form objects. We generated input polygons to obtain small sets consisting of non-intersecting polygons. A subset of these polygons are attached to other polygons belonging to the same set through a common vertex. The idea is to model human like figures with polygons where each polygon corresponds to a portion of the body. We group these polygons into sets. These sets become the basis of objects that form the building blocks of the hierarchy.

The algorithm for computing the separation between two objects was tested for various cases starting at the root of the hierarchy tree and proceeding to level i, for $i = 1, \ldots, 6$. Each level i corresponds to the maximum depth for which $H_i(O_1)$ and $H_i(O_2)$ are considered to find the separating line between OBSP's O_1 & O_2.

As expected, the fragments reduced as we considered increasing levels in the tree hierarchy during the computation of the separating line, i.e. when we considered the object as a union of a number of ellipses instead of one ellipse. However this was accompanied by a nominal increase in the time complexity of the procedure as measured by the time required to insert polygons. The advantages of reduction of fragmentation are substantial since the time of construction of the BSP is not a bottleneck in graphic modeling and hidden surface removal.

The results are graphically presented in figure 5 and figure 6.

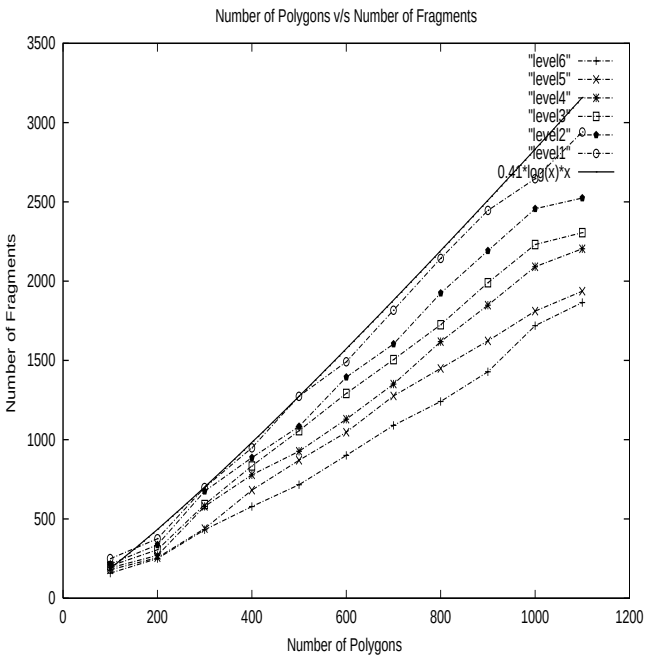

Figure 5: Number of Fragments generated v/s Polygons Inserted

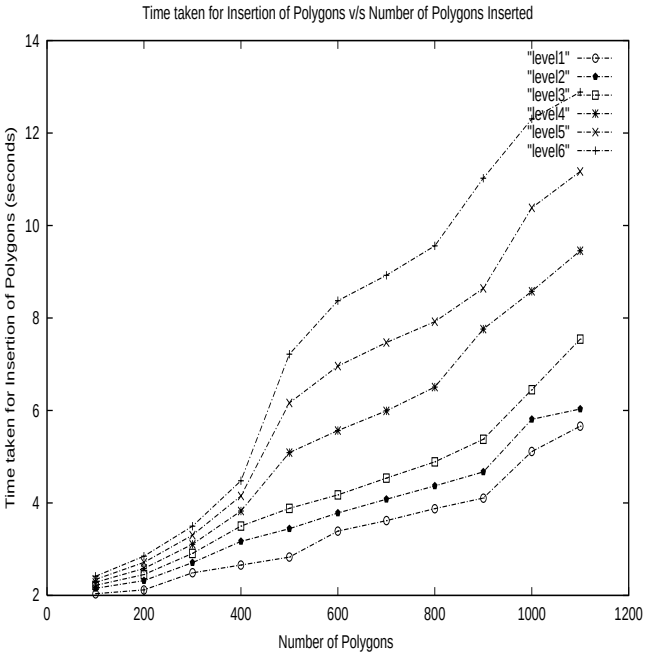

Figure 6: Time taken for Insertions v/s Number of Polygons

5 Conclusion

We have presented implementation results for an algorithm for BSP construction which instead of using polygons only deals with the objects as an entity. We have presented ways of composing objects. This allows us to insert new polygonal objects while maintaining properties which are beneficial in deleting objects from the scene or in simulating motion in the scene. These properties are critically a separation of objects and polygons and maintenance of fragments at the leaves which enables constant time fragmentation of the scene. When motion of an object takes place, instead of deleting each polygon and reinserting them, we apply the motion on the BSP Tree of the object as a whole. Hence, we are reducing the effort of calculating the BSP structure and the fragmentation involved in the construction of the the BSP Tree of the object at the cost of finding separating lines between objects.

Moreover, the object based hierarchy has been used for effectively computing separation of objects by decomposing the ellipse cover of the object into a union of ellipse covers as naturally generated by the object hierarchy. This decomposition improves the probability of finding a separating line between the polygons in the two objects. This has been shown to reduce the number of fragments of the polygons. The rendering time of a dynamic scene is greatly reduced when we have a lesser number of fragments. In the real world, we generally do not find objects so close to each other that we cannot find separating planes between them. This has important consequences in solid modeling and hidden surface removal where the complexity is determined by the number of fragments. Another advantage as a consequence is the object-based hierarchy. This allows us composition and decomposition of objects in an effective manner. Since the BSP for each object and each sub-object is constructed during composition, decomposition is readily achieved. We believe that as scenes grow more and more complex, the reduction in the fragmentation advocated by our algorithm and the consequent time complexity would be much better. Thus, our algorithm will give better result for most real world views.

References

[1] C. L. Bajaj, V. Pascucci, A. Shamir, R. Holt, A. Netravali., Multiresolution Molecular Shapes". TICAM (UTexas Austin) report, 99-42 (1999).

[2] Chrysanthou, Y., and Slater, M., Computing dynamic changes to BSP trees, Computer Graphics Forum (EUROGRAPHICS '92 Proceedings), 11(3), 321–332, sep 1992.

[3] Hubbard, P.M. Approximating polyhedra with spheres for time critical collision detection, ACM Transactions on Graphics 15, 3 (1996).

[4] Naylor, B., Amanatides, J., and Thibault, W., Merging BSP Trees Yields Polyhedral Set Operations, Computer Graphics (SIGGRAPH '90 Proceedings), 24(4), 115–124, aug 1990.

[5] Naylor, B., Interactive solid geometry via partitioning trees, Proceedings of Graphics Interface '92, 11–18, may 1992.

[6] Naylor, B., Partitioning tree image representation and generation from 3D geometric models, Proceedings of Graphics Interface '92, 201–212, may 1992.

[7] Naylor, B., SCULPT An Interactive Solid Modeling Tool, Proceedings of Graphics Interface '90, 138–148, may 1990.

[8] Paterson, M., and Yao, F., Efficient Binary Space Partitions for Hidden-Surface Removal and Solid Modeling, Discrete and Computational Geometry, 5(5), 485–503, 1990.

[9] Sharir, M., and Overmars M., A Simple Output-Sensitive Algorithm for Hidden Surface Removal,Dec '89

[10] A.Kumar ,V. Kwatra, B. Singh, S. Kapoor , Separating Plane BSP for Hidden Surface Removal,, *ICVGIP'98.*

[11] A.Kumar ,V. Kwatra, B. Singh, S. Kapoor , Using Separating Planes between Objects for Efficient Dynamic Space Partitioning,, *ICVC'99.*