On Constructing Binary Space Partitioning Trees

Ravinder Krishnaswamy
Auto Trol Technology
Research and Development
12500 North Washington
Denver, CO 80233

Ghasem S. Alijani
Shyh-Chang Su
Computer Science Department
University of Wyoming
Laramie, WY 82071

Abstract

Binary Space Partitioning Trees have several applications in computer graphics. We prove that there exist n-polygon problem instances with an $O(n^2)$ lower bound on tree size. We also show that a greedy algorithm may result in constructing a tree with $O(n^2)$ nodes, while there exist a tree for the same n-polygon instance with only O(n) nodes. Finally, we formulate six different heuristics and test their performance.

1. Introduction

The hidden line elimination problem is of fundamental importance in computer graphics [4,6,7]. The Binary Space Partitioning (BSP) tree is one approach to the problem [1], and has received recent attention in [2,3,8]. An attractive aspect of the BSP tree is that once the tree corresponding to a collection of objects is created, the scene represented by the tree can be displayed with hidden line elimination in time that is linear in the number of nodes of the tree. This efficiency inherent in BSP tree based hidden line elimination is cause for serious consideration of BSP trees to be used in parallel real-time scene generation [5].

In general, the tree representing a set of polygons is not unique, in fact, the number of nodes in trees representing the same scene may vary substantially depending on heuristics used in constructing the tree. The focus of this research is to evaluate several heuristics used to construct BSP trees according to the following two criteria :

(a) The size (number of nodes) of the tree.

(b) The extent to which the tree is height-balanced.

The order in which polygons in the left subtree of a node are processed is independent of the order in which polygons of the right subtree of the node are processed. This observation is the motivation for including (b) as a method of judging the potential for parallelism (balanced subdivision of labour) reflected in the tree structure.

2. Definition and Notation

Definition : A BSP tree is a binary tree constructed from a polygon list. The basic notion is that given a plane in a three dimensional scene and a viewing point, no polygon on the view point side of the plane can be obstructed by any polygon on the far side. The tree can be recursively constructed as follows :

A polygon is selected from the list and placed at the root. Each remaining polygon is tested to see which side of the plane containing the root polygon it lies in, and is placed in the appropriate side list. A polygon that intersects the plane containing the root polygon is split, and each of the pieces is placed in the appropriate list depending on which halfspace it lies in. The left and right subtrees are recursively constructed using the descendent sublists generated. Figure 1 shows a 2-D example of the above procedure.

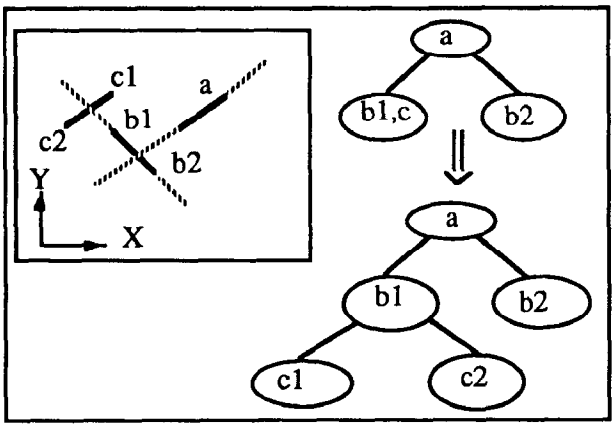

Figure 1

Notation : Let BSP(I) denote a BSP tree corresponding to an instance I of n polygons and $BSP(I)^j$ denote the tree obtained from instance I through using heuristic j.We say that BSP(I)* is a 'node-optimal' tree if BSP(I)* contains the least number of nodes of all possible trees BSP(I). 'Height-optimal' is similarly defined. It is observed in [1] that a BSP tree may contain as much as $O(n^2)$ nodes, where n = | I |. We show that in fact the node-optimal tree BSP(I)* may contain as many nodes too.

Bounds on Tree Size

Theorem 1: There exists an instance I such that BSP(I)* contains $O(n^2)$ nodes.

Proof : Let I_1 denote the set of polygons $P_0, P_1 ..., P_{(n/2\ -1)}$ obtained as follows. P_0 is a rectangle with coordinates <(0, 0, 1),(1, 0, 1),(1, n/2+2,1),(0, n/2+2,1)>, and p_i is obtained by translating p_{i-1} unit distance along the positive z direction for $1 \leq i < n/2$.

Let I_2 denote the set of polygons $P_{n/2}, P_{n/2+1}, ..., P_{n-1}$ where $P_{n/2}$ is a rectangle with coordinates <(2,1,0), (3,1,0), (3,1,n/2+1), (2,1, n/2+1)>, and p_i is obtained by translating P_{i-1} unit along the y direction for $n/2 \leq i < n$. (see Figure 2).

Let I = I1 U I2. We claim that any tree constructed from I must contain exactly n(n+4) /4 nodes.

Let $l_{i, n/2+j}$ represent the line of intersection of the planes containing the polygons P_i and $P_{n/2+j}$ where $0 \leq i,j < n/2$.

Let BSP $(I)^*$ be a 'node-optimal' tree and let P be some node in the tree. Without loss of generality assume that P was obtained from P_i for some $i < n/2$.

We say the line $l_{i, n/2+j}$ is properly contained in P if $l_{i, n/2+j}$ intersects the interior of P.

Let k lines be properly contained in P. This means that polygons $P_{n/2+j1}$, $P_{n/2+j2}$,.., $P_{n/2+jk}$ were split by P when P was selected to be the root of some subtree. Since each line $l_{i, n/2+j}$ is properly contained in some polygon of the tree, the number of such splits equal to the number of lines $l_{i,j} = (n/2)^2$. Therefore, the number of polygons in the tree is equal to $(n/2)^2 + n = n(n+4)/4$.

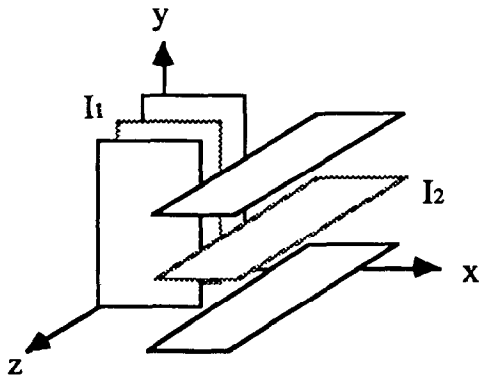

Figure 2

Several different heuristics have been formulated and applied in constructing the BSP tree [1,3]. Each of these heuristics perform near optimal under a certain well-defined conditions. In the following we refer to a greedy algorithm as the heuristic which at each step selects a polygon as the root that intersects the least number of polygons. In the next theorem we in fact show that a greedy algorithm results in the worst case tree ($BSP(I)^g$) for even a fairly straightforward problem instance. This further justifies the need to formulate different heuristics and carefully examine their performance.

Theorem 2 : There exists an instance I where:

$$\frac{|BSP(I)^g|}{|BSP(I)^*|} = O(n)$$

Proof: Consider the instance of polygons I as in Theorem 1. Augment this instance with a polygon P with vertex coordinates < (1-a, n/2+3, 0) , (1-a, n/2+3, n/2 +3), (1-b, n/2, n/2+4), (1-b, n/2+3, 0)> for constants a and b that cause P to intersect the original set $I_1 \cup I_2$ so that :

(a) The plane containing P intersects all n/2 polygons in I_1.

(b) The plane containing P intersects at least one and no more than k_1 polygons of I_2 for some constant k_1.

(c) Each of the pieces of the polygons of I_1 that lie in the same halfspace defined by the plane containing P as the $n/2-k_1$ polygons of I_2 that do not intersect the plane containing P, intersect at most k_2 of these $n/2-k_1$ polygons in I_2 for some constant k_2 (see Figure 2).

Augment I_1 by polygon P' with coordinates <(0,0,0.5), (1,0,0.5), (1,n/2-1,0.5), (0,n/2-1,0.5)>. Note that the problem instance size is n + 2, and the planes containing polygons from I_1 intersects n/2 polygons from I_2 and P, (i.e. each plane intersects a total of n/2+1 polygons). Each of the planes containing polygons from I_2 intersects n/2 polygons from I_1, and polygon P', except for the polygon from I_2 with the maximum y coordinates, which intersects only the n/2 polygons from I_1. The plane containing P' intersects n/2 polygons from I_2 and P, and the plane containing P intersects K_1 polygons from I_2 and n/2 polygons from I_1.

The greedy algorithm would start by selecting the topmost polygon from I_2 as the root, creating as one subproblem polygons from $I_1 \cup P'$ and I_2 which is basically the same situation as the instance in Theorem 1. We already know from Theorem 1 that the subtree corresponding to this problem instance must contain $O(n^2)$ nodes. Thus the greedy algorithm would create a tree with $O(n^2)$ nodes.

The optimal method of constructing the tree would select p as the root, and create $n/2+k_1$ new polygons at the first level. Thereafter, the polygon in each partition intersecting the least number of other polygons in the same partition will intersect at most k_1 or k_2 polygons. The number of nodes in the tree will therefore be no more than $n+ (k_1 + k_2 +1) * n/2 +1$.

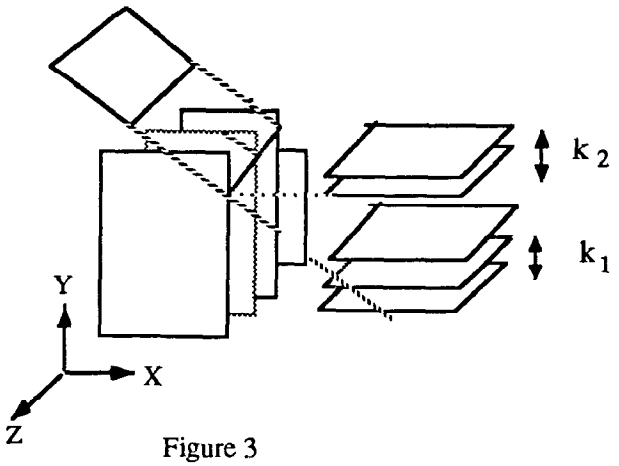

Figure 3

In the following section, we formulate and evaluate six different heuristics in order to explore their potential for parallel computation and real-time scene generation. The formulation process benefits from inherent properties of a directed graph corresponding to the subproblem. While the evaluation is established based on the criteria which stated earlier.

3. Evaluating Heuristics

Thiabult in [3] examines three different schemes to construct the BSP tree. Briefly, the idea of his heuristic is to randomly select a constant size subset of polygons and test each member of this set against the remaining polygons to compute the heuristic 'value'. The one that minimizes the heuristic is selected to be the splitting node.

Our approach differs from the above method in the following important way. The candidate set size in our method is varying and is in fact equal to the size of the subproblem. Furthermore, in addition to the geometric relationships on polygons in a subproblem, we try to apply the combinational properties of the subproblem by analyzing the *directed graph* corresponding to the subproblem. The directed graph is obtained by viewing polygons as nodes and a directed edge exists between nodes **a** and **b** iff the plane containing polygon **a** intersects polygon **b**. The inherent properties of the directed graph provide a significant flexibility for constructing the tree and analyzing the instances of the problem. The main steps in all heuristics can be stated as follows:

1. From the subproblem I of polygons, construct a directed graph, G(I).
2. Extract combinatorial information from G(I).
3. Extract geometric information from I.
4. Use data from steps 2 and 3 to select the candidate polygon, P, as root.

Considering these major steps, six different heuristics are formulated as follows:

H-1 : Select minimum outdegree node P from G(I).

H-2 : Select maximum indegree node P from G(I).

H-3 : Select minimum outdegree node from G(I) and if there is more than one then of all such nodes, select one with maximum indegree.

H-4 : Select node P from G(I) that minimizes (| IN (P) - OUT (P) |).

H-5 : Select node P from G(I) that minimizes (Outdegree (P) * | IN (P) - OUT(P) |).

H-6 : Select node P from G(I) that minimizes (S * Outdegree + | IN (P) - OUT(P) |).

where

IN(P) is the number of polygons in halfspace which defined by the plane containing polygon P that includes the origin.

OUT(P) is the number of polygons in halfspace defined by plane which containing polygon P that does not include the origin. And S is an integer positive with value between 50-100.

The motivation for H-1 and H-3 is that we want to select nodes that introduce the least number of addition nodes into the tree. The reason for constructing the heuristic H-2 is that the two graphs resulting from the choice of P each corresponding to an induced subproblem has as few edges as possible. While the objective for establishing the heuristic H-4 is to find height-balanced tree in order to increase the degree of parallelism and reduce the complexity of subproblems. The heuristics H-5 and H-6 are similar to the heuristics in [3] except that we select the node P from a global set of candidate nodes.

3.1. Experimental Results

A set of experiments is conducted to evaluate the performance each of these heuristics under different conditions. Figure 4 shows the performance of these heuristics in terms of number of nodes as well as the height of the tree. The environment in which these heuristics have been tested contains several solid objects such as cylinder, cone and simple blocks. These objects are divided into a set of polygons with cardinality of 126. As the result indicate H-1 and H-3 tend to minimize the number of nodes while the heuristics H-5 and H-6 minimize the height of the tree. It should be noted that the number of nodes generated by heuristics H-4 , H-5 and H-6 is substantially more compared with H-1, H-2 and H-3. However, the difference in the heights is not significant.

Heuristics	H-1	H-2	H-3	H-4	H-5	H-6
Number of Nodes	212	256	210	793	463	284
Height	39	43	40	38	37	37

Figure 4

In order to investigate the effectiveness of these heuristics in more general case, we conduct the next set of experiments based on different number of randomly generated polygons. Figure 5 shows the performance of the heuristics when the number of polygons is selected to be 100. While figure 6 shows the same type of experiment using 50 randomly generated polygons. The result indicate that the heuristics H-4 and H-5 produce trees with minimum height. More important, the heuristic H-2 tend to minimize the number of nodes as well as the height of the tree. Since this heuristic selects a node from the directed graph with maximum indegree. This process reduces the height and the number of nodes significantly.

Heuristics	H-1	H-2	H-3	H-4	H-5	H-6
Number of Nodes	1509	1437	1514	1693	1593	1508
Height	86	18	87	14	14	34

Figure 5

Heuristics	H-1	H-2	H-3	H-4	H-5	H-6
Number of Nodes	501	475	494	509	496	507
Height	45	15	42	12	11	23

Figure 6

Finally, we conduct a test which represents the worst-case model. In this case a set of polygons is divided into three subsets each containing a number of polygons. Polygons in each subset are intentionally constructed in such a way that they intersect with each other in a cyclic fashion . In other words, each polygon intersects a polygon or it will be intersected by a polygon. As the results indicate (see figure 7) more complex heuristics such as H-4, H-5 and H-6 cannot take the advantage of this instance. However, in this particular situation, a simple heuristic such as H-1 may perform better in terms of number of node as well as the height.

Heuristics	H-1	H-2	H-3	H-4	H-5	H-6
Number of Nodes	108	145	112	255	237	154
Height	30	29	32	26	26	31

Figure 7

In fact it can be shown that for certain class of instances the heuristics H-1 and H-3 produce an optimal number of nodes.

Theorem 3: If G(I) in an Acyclic Directed Graph, then $| BSP (I)^1 | = | BSP (I)^3 | = | BSP (I)^* |$.

Proof: If G(I) is a DAG then there exist at least one node with outdegree equal zero. Then the heuristics H-1 and H-3 select such a node to be the root. This does not introduce any more polygons since the subproblems created by such a selection also represent DAGs, therefore H-1 and H-3 will create tree with | I | nodes.

4. Conclusion

The work reported in this paper is part of ongoing research concerning the practicality of the use of BSP trees for sequential and parallel hidden line elimination.

We compared the performance of different heuristics for BSP tree construction with respect to tree height and size. From the theorem in section 2 and the tables of results our conclusion is that the performance of heuristics varies widely according to the family of graphs representing the family of scene instances. Thus, a priori knowledge of scene characteristics world be a strong influencing factor on heuristic choice. The heuristics H-4 and H-5 consistently generated trees of low height, suggesting their utility for parallel BSP tree processing.

Future work includes designing more elaborate heuristics that include breaking cycles in the directed graph by examining directed path of bounded length from a candidate mode. A parallel implementation of BSP tree construction is currently underway on a sixteen node mesh connected transputer based system.

References

[1] H. Fuchs, G. D. Abram and Eric D. Grant, "Near Real-Time Shaded Display of Rigid Objects", *Computer Graphics, Vol. 17, No. 3*, July 1983.

[2] W. C. Thibault and Bruce F. Naylor, " Set Operations on Polyhedra Using Binary Space Partitioning Trees", *Computer Graphics, Vol. 21, No. 4*, July 1987.

[3] W. C. Thibault, "Application of Binary Space Partitioning Trees to Geometric Modeling and Ray-Tracing", Ph.D Thesis *Georgia Institute of Technology*, September 1987.

[4] J.D. Foley and A. Van Dam, "Fundamentals of Interactive Computer Graphics", *Addison-Wesley*, 1982.

[5] P. Atkin and J. Parker, "High Performance Graphics with the IMS T800", INMOS Technical Note 37, March 1988.

[6] I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms", *Computing Surveys, Vol. 6, No. 1*, 1974.

[7] P.R. Atherton, "A Scan-Line hidden surface Removal Procedure for Constructive Solid Geometry.", *Computer Graphics, Vol. 17, No. 3*, pp. 73-82, July 1983.

[8] Norman Chain and Steven Feiner, "Near Real-Time Shadow Generation Using BSP Trees", Computer Graphics, Vol. 23, No. 3, July 1989.