View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by NASA Technical Reports Server

341

N90-20657

Binary Space Partitioning Trees and Their Uses

By: Bradley N. Bell

Barrios Technology Inc.

1331 Gemini

Houston, Texas 77058

ABSTRACT

Binary Space Partitioning (BSP) Trees have some qualities that make them useful in solving many graphics related problems. The purpose of this paper is to describe what a BSP tree is, and how it can be used to solve the problem of hidden surface removal, and constructive solid geometry. The BSP tree is based on the idea that a plane acting as a divider subdivides space into two parts with one being on the positive side and the other on the negative. A polygonal solid is then represented as the volume defined by the collective interior half spaces of the solid's bounding surfaces. The nature of how the tree is organized lends it self well for sorting polygons relative to an arbitrary point in 3 space. The speed at which the tree can be traversed for depth sorting is fast enough to provide hidden surface removal at interactive speeds. The fact that a BSP tree actually represents a polygonal solid as a bounded volume also makes it quite useful in performing the boolean operations used in constructive solid geometry. Do to the nature of the BSP tree polygons can be classified as they are subdivided. The ability to classify polygons as they are subdivided can enhance the simplicity of implementing constructive solid geometry.

INTRODUCTION

The goal of this paper is explain what a Binary Space Partitioning (BSP) tree is and how it can be used to depth sort polygons and perform boolean operations on polyhyedra. Depth sorting of polygons is a technique that has been widely used on personal computers to provide hidden surface removal. With the use of a BSP tree polygons can be sorted fast enough to support the interactive display of shaded polygons with hidden surfaces removed even on a personal computer. Also BSP trees can be employed to solve the problem of Constructive Solid Geometry (CSG). CSG, which is implemented in many model builders, provides the capability to describe complex objects as the intersection, union, and/or difference of simpler primitives. To understand how to use a BSP tree it is important that we have a clear idea of what one is.

BSP TREES

A Binary Space Partitioning (BSP) tree is a data structure that represents the partitioning of space where each branching node represents a plane that divides the space it occupies into two parts and each leaf represents either a polygon (for depth sorting) or a bounded volume (for boolean operations). Given any point in space polygons can be sorted far to near or near to far by using a simple but mathematically determined traversal of the tree. Boolean operations on polyhedron can be performed by cutting the polygonal representation of one operand by the BSP representation of the other.

BUILDING BSP TREE TO DEPTH SORT POLYGONS

In order to use a BSP tree to depth sort polygons the tree can be constructed by using the polygons themselves as planes that subdivide space. This can be accomplished by

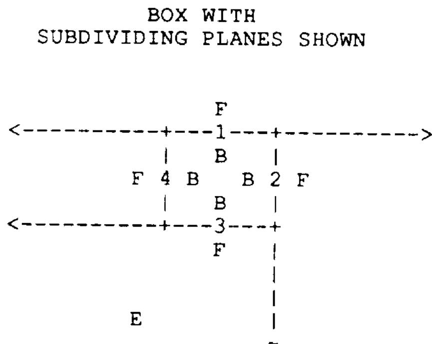
first determining the data structure needed to define a node in the tree. The following is used as an example.

```
typedef struct node
{
     float        A, B, C, D;
     POLYGON      *p_poly;
     struct node  *p_front;
     struct node  *p_back;
}
NODE;
```

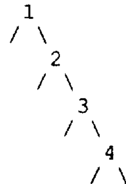
Note: It is convenient to use the sign of the value returned from the plane equation when determining if a point is in front or in back of the plane.

Here the node contains the polygon's plane equation coefficients, a pointer to the polygon representing the sub-dividing plane, and two pointers to nodes that represent the space in front and the space in back of this polygon. Well use a box to illustrate how the tree would be constructed (fig. 2).

BOX WITH
SUBDIVIDING PLANES SHOWN



BSP TREE OF BOX
FRONT<-- -->BACK



F = Front side of polygon
B = Back side of polygon
E = Example Eye point

(fig. 2)

Where the numbers are used to identify the polygons that make up the box. We'll start with a group of polygons at the root of the tree. Then selecting polygon number 1, we will divide the remaining polygons into two groups one representing the polygons in front and the other representing the polygons in back. In this example all of the polygons are on the back side of the first polygon. Next we will select a polygon from each group which will be used to subdivide its group in much the same way as we did the root of the tree, and when a group of polygons contains only one polygon then that branch of the tree is completed. Once the tree has been built a simple but mathematical traversal of the tree can be performed to determine which order to display the polygons in so that the nearest one gets drawn last.

SORTING FROM FAR TO NEAR

To begin sorting polygons from far to near start by entering the eye point into the plane equation of the root node to determine which side of the polygon the eye is on. In (fig. 2) the eye point is shown to be on the back side of polygon number 1. In order to sort the polygons from back to front the half of the tree representing the opposite side of the polygon from the eye must be traversed first then the polygon in this node then the side of the tree representing the side of the polygon the eye is on. So in this example we would traverse the Front side of the tree starting at the root before we would output polygon number 1 after which we would traverse the Back side of the tree. As we traverse the tree we perform the same eye plane test as was done before but using the plane equation at the node we are on in the tree to determine which branch of the node will be traversed first. The tree traversal proceedure can easily be implemented as a recurrsive function (fig. 3 as an example).

SORTING FROM NEAR TO FAR

To sort polygons from near to far the process is identical except instead of traversing the side of the polygon that is on the opposite side of the dividing plane first, you traverse the side of the polygon that the eye is on first.

ORIGINAL PAGE IS
OF POOR QUALITY

```
FarToNear( n, x, y, z )
NODE *n;
float x, y, z;
{
  float p;

  if( n )
  {
    p = n->A * x + n->B * y
      + n->C * z + n->D;

    /* ASSUMING THE NORMAL OF
       THE PLANE IS POINTING
       TO FRONT HALF */

    if( p < 0.0 )
       FarToNear(n->p_front,x,y,z);
    else
       FarToNear(p->p_back,x,y,z);

    DrawPolygon(n->p_poly);

    if( p > 0.0 )
       FarToNear(n->p_front,x,y,z);
    else
       FarToNear(n->p_back,x,y,z);
  }
}
```

(fig. 3)

BOOLEAN OPERATIONS ON POLYGONAL MODELS

One way to perform boolean operations on polygonal models is to use a BSP tree. This can be accomplished by first constructing a BSP tree representation of each model then using the tree of one model to subdivide the polygons of the other model into inside and outside components. Then depending on the operation being performed the pieces needed are gathered together from both models to form the result. The BSP representation however differs slightly from the one used to depth sort polygons.

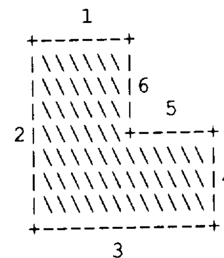
BUILDING BSP TREE TO PERFORM BOOLEAN OPERATIONS

The BSP tree used to perform boolean operations is constructed in a similar way as the one used to depth sort polygons with the exception that the branches of the tree represent the division of space into inside and outside components with the subdividing plane representing a polygon which is part of the model. In order to explain how the tree could be constructed we need to determine what type of data structure to use. The following is given as an example.

```
typedef struct node
{
     float        A, B, C, D;
     POLYGON      *p_poly;
     struct node *p_outside;
     struct node *p_inside;
}
NODE;
```

Here a node in the tree contains a plane equation, a reference to a polygon, a pointer to the branch of the tree representing the outside of the volume and a pointer to the branch representing the inside. To construct the BSP representation of the model we first select a polygon from the model that we will use as the dividing plane at the root of the tree. We then proceed to divide the remaining polygons by the dividing plane at the root of the tree into two groups, one to the outside of the plane and the other to the inside. Each group represents a branch from the root node of the tree. Next from each group a polygon is selected to become the dividing plane of its group and is placed into the appropriate node. Each group is then subdivided by its associated node and placed into two separate groups again representing the polygons to the inside and outside of the dividing polygonal plane. This proceedure is performed recurrsively until their is only one polygon left in the group which is then placed into its own node with both of its branch pointers set to 0. The following is given as an example.

DIAGRAM OF SIMPLE MODEL

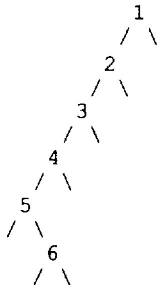


KEY:
1-6 Polygon identification
\ The interior of the model

ORIGINAL PAGE IS OF POOR QUALITY

TREE REPRESENTING THE ABOVE MODEL

INSIDE<--- --->OUTSIDE



Once the BSP tree has been constructed for both operands of the boolean operation the polygons of each model can be subdivided by the other model's BSP tree. To subdivide a polygon by the BSP tree we take the polygon and start at the root of the tree and test to see if the polygon is inside, outside, or on both sides of the subdividing plane. If the polygon is on both sides of the subdividing plane then it is split into two polygons with the one representing the inside part and the other representing the outside part. The polygon (parts) is (are) then tested against the plane in the node pointed to by the associated branch. This proceedure is performed recurrsively until a branch pointing to nothing is reached at which time the polygon (part) is given the classification of the branch. If the polygon being tested lies in the same plane as the subdividing plane then a more complex procedure is required. First we send the whole polygon down the inside branch of the tree. Next we make note of the classifications given to the resulting parts. Then we send each part down the outside branch of the tree. If the part comes back with the same classification as it did going down the inside branch of the tree then it is correctly classified. Should the part get subdivided while being passed down then the subparts that have the same classification are correctly classified. The the parts that come back with a different classification are considered as coplaner polygons and are assigned the classification of OPPOSITE if the polygon's normal points in the opposite direction as the normal of polygon it is coplaner to otherwise it is given the classification of SAME. Once all of the polygons in both operands have been classified in this manner the resultant model can be formed. The following table describes for each boolean operator which polygons are taken from each operand to form the resulting model.

OPERATION	AND		OR		-	
OPERANDS	A	B	A	B	A	B
INSIDE	X	X				F
OUTSIDE			X	X	X	
OPPOSITE					X	
SAME	X		X			

= Do not use to form result

X = Use directly to form result

F = Flip normal of polygon before using to form result

CONCLUSION

Even though Binary Space Partitioning (BSP) trees can be used to perform the tasks described in this paper they are not practical when working with models that are highly complex. The tree tends to grow exponentially as the model complexity grows linearly. However they do offer implementation simplicity and therefore have a useful place in software development.

ORIGINAL PAGE IS OF POOR QUALITY