Set Operations on Polyhedra Using Binary Space Partitioning Trees

Vikram Chandrashekhar, Group 16

Project Overview

One in 323 children in the US are born with cerebral palsy. Two out of every three children born with cerebral palsy could walk if they had proper orthoses to alleviate their condition. Fusiform Medical Devices has developed a process to reduce waste, reduce time and increase efficiency of orthosis design and fabrication. Currently, the process requires approximately 10 hours to create a single custom-designed orthosis in SolidWorks. The goal of our project is to develop a browser-based constructive solid geometry application for 3D anatomical models. This application would allow the orthotist to add pre-designed orthotic components onto a 3D leg scan of cerebral palsy patients. In addition, this software would be much simpler to use, reducing the amount of time required to design the casts.

Paper Selection

Since the goal of our project is to develop a constructive solid geometry (CSG) application that runs in realtime, the paper chosen is “Set Operations on Polyhedra Using Binary Space Partitioning Trees” by William Thibault and Bruce Naylor. This paper provides a mathematical foundation for and algorithm for performing constructive solid geometry.

Problem Summary and Key Results

The goal of this paper is to introduce a method to efficiently perform CSG on any d-dimensional polyhedra. Methods available at the time this paper were written used boundary representations (b-reps) to represent polyhedra. B-reps are methods for representing shapes using limits. This paper proposed to use binary space partitioning trees (BSP trees) to represent and perform operations on these polyhedra more efficiently. Using BSP trees, this paper describes an algorithm to incrementally perform CSG operations including union, intersect, and subtract on d-dimensional polyhedra. The algorithm in the paper speeds up these operations and also introduces methods to optimize the memory usage of the BSP tree.

Significance of Result

The results of this paper are highly relevant to this project; we are currently working with very large triangular meshes and need to perform CSG in realtime so it is imperative that our memory usage and processing time are very low to minimize lag. This paper covered BSP tree reduction which could potentially reduce the memory and run time associated with objects created using CSG. These improvements could be particularly useful when creating objects with ~50000 vertices and ~90000 faces.

Glossary

Boundary representation (B-rep) - A d-dimensional solid represented as a collection of (d-1)-polyhedra (also called faces) represented by (d-2)-polyhedra until d = 0

Binary space partitioning tree (BSPT) - a binary tree whose non-leaf nodes are labeled with hyperplanes and whose leaf nodes correspond to cells of a partitioned d-space.

Cell - area enclosed by splitting hyperplanes

Hyperplane - a (d-1)-dimensional subspace in d-space (e.g. a 2D plane in 3D space but generalized to any higher-dimensional space)

Half-space - either of the 2 parts into which a (hyper)plane divides a d-dimensional space

R(v) - the intersection of open half-spaces from the root to any node *v*

Sub-hyperplane - the intersection of the splitting hyperplane and the region R(v) defined by any node *v*

Regular set - set that consists of its interior and its boundary

“in” - a cell is in the interior of a polyhedron

“out” - a cell is in the exterior of a polyhedron

“on” - a point is on the boundary of a polyhedron

CSG tree - a CSG representation of a set *S* in a binary tree in which the internal nodes represent set operations and the leaves are primitive polyhedra

Description of Results

Generic BSP trees represent recursive, hierarchical partitioning of d-dimensional space. Specifically, this project is concerned with three-dimensional space. BSP trees are generated by choosing hyperplanes that partition the interior of the subspace, producing two new subspaces that can be subdivided. the left and right children of the nodes of the BSP trees represent the space to the left (behind) and to the right (in front) of a splitting hyperplane respectively. The right side is taken to be the side of the hyperplane to which normal vector points. An example partitioning of 2D space is shown in Figure BSPT.

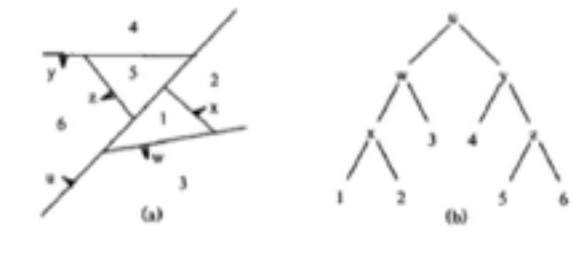


Figure BSPT. Geometry of a 2D partitioning (a) and its BSP tree (b).

Given this understanding of BSP trees, one can use the algorithm described in Figure BUILD-BSPT to convert a boundary representation of a polyhedron to a BSP tree representation. This algorithm has two requirements:

- all points on the boundary of polyhedra must lie on sub-hyperplanes of the BSP tree

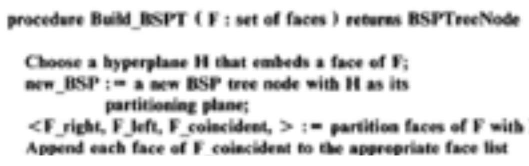
- all cells must correctly be classified “in” or “out”.

In order to ensure that all boundary points of the polyhedra lie on sub-hyperplanes of the BSP tree, the splitting hyperplanes chosen will embed faces. Using these rules, the steps required in the conversion from b-rep to BSP tree are as follows:

1. Choose a splitting hyperplane that embeds a face of the polyhedron
2. Store this hyperplane in the current node. Determine whether the other faces of the polyhedron lie to the left or to the right of the current hyperplane and pass this information to the respective subtrees.
3. If either the left or right side face list is empty, then that region is homogenous. Based on the rules in Figure BUILD-BPST, the appropriately labeled “in” or “out” node is appended to the left or right subtree depending on which list is empty.
4. This process is recursively applied to the left and right subtrees until both the left and right subtrees are empty.

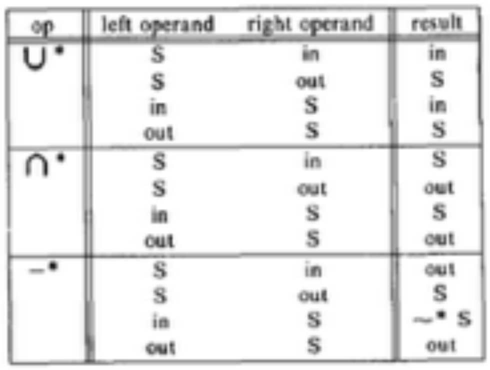
From this algorithm, the ability to insert a single face into a BSP tree follows. First the face is split by the hyperplane in the root node; the left portion of the face is passed to the left subtree and the right portion of the face is passed to the right subtree. This is repeated recursively until new leaf nodes are created with correct classification of the cells.

This paper also discusses an algorithm in Figure INCREMENTAL SET-OP to perform regularized set operations (as in Figure SIMPLIFY) on B-reps and BSP trees. Given BSP tree T’ representing polyhedron T and B-rep (or BSPT) B’ representing polyhedron B, the algorithm in order to perform these set operations is as follows:



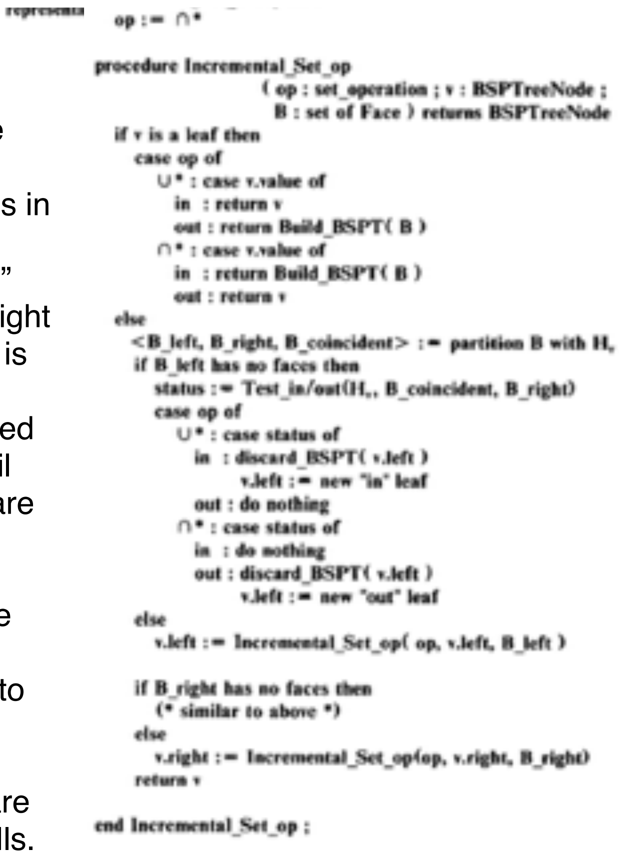
```
procedure Build_BSPT ( F : set of faces ) returns BSPTreeNode

  Choose a hyperplane H that embeds a face of F;
  new_BSP := a new BSP tree node with H as its
             partitioning plane;
  <F_right, F_left, F_coincident, > := partition faces of F with H;
  Append each face of F_coincident to the appropriate face list
```



op	left operand	right operand	result
$\cup^*$	S	in	in
	S	out	S
	in	S	in
	out	S	S
$\cap^*$	S	in	S
	S	out	out
	in	S	S
	out	S	out
$-^*$	S	in	out
	S	out	S
	in	S	$\sim^* S$
	out	S	out

Figure SIMPLIFY. Expression simplification rules. S is an arbitrary regular set.



representa

```
   op := ∩*

procedure Incremental_Set_op
                  ( op : set_operation ; v : BSPTreeNode ;
                    B : set of Face ) returns BSPTreeNode
   if v is a leaf then
      case op of
         ∪* : case v.value of
            in  : return v
            out : return Build_BSPT( B )
         ∩* : case v.value of
            in  : return Build_BSPT( B )
            out : return v
   else
      <B_left, B_right, B_coincident> := partition B with H,
      if B_left has no faces then
         status := Test_in/out(H,, B_coincident, B_right)
         case op of
            ∪* : case status of
               in  : discard_BSPT( v.left )
                     v.left := new "in" leaf
               out : do nothing
            ∩* : case status of
               in  : do nothing
               out : discard_BSPT( v.left )
                     v.left := new "out" leaf
      else
         v.left := Incremental_Set_op( op, v.left, B_left )

      if B_right has no faces then
         (* similar to above *)
      else
         v.right := Incremental_Set_op(op, v.right, B_right)
      return v

end Incremental_Set_op ;
```

Figure INCREMENTAL SET-OP. Psuedo code for the incremental set evaluation algorithm.

1. Insert all faces of B’ into BSP tree T’
2. If at some node *v*, no part of B’ is found to lie on one side of the splitting hyperplane then that region is homogenous
3. Determine whether the region is “in” or “out” of B
4. Determine what to do with the appropriate subtree by following the control flow in Figure INCREMENTAL SET-OP
5. If *v* is a leaf then R(*v*) is homogenous and will either retain T’s value or B’s value

The paper also presented a similar algorithm to perform on CSG trees in order to convert them to BSP trees. However, this algorithm was less significant for this project as CSG trees are much more useful visually than computationally.

However, this paper did also include BSP tree reduction as a way to reduce the amount of memory required to represent these complex polyhedra. There are 2 cases in which it is possible to reduce the BSP tree:

1. Both subtrees a given node *v* are cells with identical values
2. A node that has one child cell and no part of the boundary in its sub-hyperplane

In the first case, the tree can be reduced by eliminating that node and replacing it with either an “in” or “out” node as appropriate. The solution to the second case is to eliminate the node altogether.

Conclusions

BSP trees serve as a unifying data structure to perform search operations, set operations, and visible surface determination whereas a b-rep requires separate structures for all of those computations. This reduces the conceptual and computational complexity associated with CSG operations and search operations.

Assessments

I think overall, this paper was very good. It was very detailed in all aspects of the mathematical basis for the BSP trees. In addition, these explanations were very easy to follow, even to someone who had no knowledge in the field of graphics. The pseudocode is also very clear and provides a clear picture of how to implement these algorithms. However, while these algorithms are very clear, they are restricted to linear approximations of non-linear polyhedra like spheres.

References

1. Thibault, W. C., & Naylor, B. F. (1987). Set operations on polyhedra using binary space partitioning trees. ACM SIGGRAPH Computer Graphics SIGGRAPH Comput. Graph., 21(4), 153-162. doi:10.1145/37402.37421