Set Operations on Polyhedra using Binary Space Partitioning Trees

Vikram Chandrashekar
Group 16
Project Summary

Browser Based Constructive Solid Geometry for Anatomical Models

- Orthoses for cerebral palsy patients
- Fusiform developed a process to reduce waste, reduce time and increase efficiency of orthotic design/fabrication
- Currently: ~10 hour process to create orthotic in SolidWorks
- Browser based software to add pre-designed orthotic components

Goal of Paper:
- Use binary space partitioning trees (BSPTs) to perform constructive solid geometry (CSG) operations
  - Why: BSPTs are much faster and more unified than other methods to compute CSG operations

Application to Project:
- Modify current CSG package to optimize nodes of BSPT
Some terms and Definitions

- **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
BSPT terminology

- Each internal node $v$ of BSPT represents region of space $R(v)$
- $R(v)$ is the intersection of open halfspaces on the path from the root to $v$
- Associated with partitioning hyperplane $H_v$
- 3 regions
  - $R(v) \cap H_v^+$
  - $R(v) \cap H_v^-$
  - $R(v) \cap H_v$
- Sub-hyperplane ($SHp(v)$) - $R(v) \cap H_v$

More formally, for a hyperplane

$$H = \{(x_1, \ldots, x_d) | a_1 x_1 + \cdots + a_d x_d + a_{d+1} = 0\},$$

the right (or in B-rep parlance, the "front") halfspace of $H$ is

$$H^+ = \{(x_1, \ldots, x_d) | a_1 x_1 + \cdots + a_d x_d + a_{d+1} > 0\},$$

and the left (or "back") halfspace of $H$ is

$$H^- = \{(x_1, \ldots, x_d) | a_1 x_1 + \cdots + a_d x_d + a_{d+1} < 0\}.$$
**Generic BSPTs**

- Recursive, hierarchical partitioning of d-dimensional space
- Nodes store splitting hyperplanes
- Distinction between halfspaces determined by normal vector - arbitrary choice
- Right subtree - region lying on side pointed to by normal
- Left subtree - the other region

*Figure BSPT. Geometry of a 2D partitioning (a) and its BSP tree (b).*
B-rep -> BSPT

- Requirements
  - All points on boundary of polyhedra lie in sub-hyperplanes of the resulting tree
    - Embed faces
  - Correct classification of cells
    - “In” vs “out”

- Algorithm
  - Choose hyperplane \( H \)
  - Partition faces left of, right of, or coincident with \( H \)
    - When empty, we know that region is homogenous
  - Recursively apply to left and right face subtrees

```plaintext
procedure Build_BSPT ( F : set of faces ) returns BSPTTreeNode

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 \( F_{\text{coincident}} \) to the appropriate face list of new_BSP;

if (F_left is empty) then
  if (F_coincident has the same orientation as \( H \)) then
    (* faces point "outward" *)
      new_BSP.left := "in";
  else
    new_BSP.left := "out";
  else
    new_BSP.left := Build_BSPT( F_left );

if (F_right is empty) then
  if (F_coincident has the same orientation as \( H \)) then
    new_BSP.right := "out";
  else
    new_BSP.right := "in";
  else
    new_BSP.right := Build_BSPT( F_right );

return new_BSP;
end; (* Build_BSPT *)
```
Inserting a face

1. Let $v$ be some node in the tree (initially equal to root) and $f$ be some face to add
2. Partition $f$ by $H_v$ and pass the part of $f$ lying to the left of $H_v$ to $v$.left and part of $f$ lying to the right to $v$.right
3. Repeat this process until part of $f$ reaches a leaf (create a new node)

Using this process one can go from a trivial BSP to BSP tree representing polyhedra
Evaluating Set Operations

- **Regular set** - set that consists of its interior and its boundary
- Partition space into regions such that at least one operand is homogenous in each region (e.g. $\text{ext}(S)$ or $\text{int}(S)$)

<table>
<thead>
<tr>
<th>op</th>
<th>left operand</th>
<th>right operand</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cup^*$</td>
<td>S</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>out</td>
<td>S</td>
</tr>
<tr>
<td>in</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>$\cap^*$</td>
<td>S</td>
<td>in</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>out</td>
<td>out</td>
</tr>
<tr>
<td>in</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>$-^*$</td>
<td>S</td>
<td>in</td>
<td>out</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>out</td>
<td>S</td>
</tr>
<tr>
<td>in</td>
<td>S</td>
<td>out</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>S</td>
<td>out</td>
<td></td>
</tr>
</tbody>
</table>

Figure SIMPLIFY. Expression simplification rules. $S$ is an arbitrary regular set.
Evaluating Set Operations

Given BSP tree $T'$ representing polyhedron $T$ and B-rep (or BSPT) $B'$ representing polyhedron $B$.
Perform $T -* B$:

1. Insert all faces of $B'$ into $T'$
2. If at some node $v$, no part of $B'$ is found to lie on one side of $H_v$ (let’s say left) then $R(v.left)$ is homogenous
3. Determine whether the region is “in” or “out” of $B$
Evaluating Set Operations

4. Determine what to do with the appropriate subtree (v.right or v.left) given the operation and type of region

5. If v is a leaf, then R(v) is homogenous and will either retain T’s value or B’s value
Evaluating Set Operations

- Perform $T \ast B$

Figure SET-OP. BSP tree $\ast$ B-rep $\rightarrow$ BSP tree.
CSG Trees

- A binary tree in which the internal nodes represent (regularized) set operations and leaves are instanced primitives
- Easier visual representation for complex objects
- Not particularly useful computationally (need to convert to BSPT)
BSP Tree Reduction

- Eliminate certain nodes without changing the set - reduction in memory
- Both subtrees of node $v$ are cells with identical values
  - Replace subtree with single value
- Node that has one child and contains no part of the boundary ($u$)
  - Remove this node

Figure REDUCE. Nodes $u$ and $z$ can be eliminated.
Conclusions

- Similarity between octrees and BSP trees
  - Recursively subdivide space
  - Assign values to leaves
  - Dimension independent

- Key difference: BSPT hyperplanes do not have to be axis-aligned
  - Octrees tend to be more verbose as a result (more memory)

- B-rep algorithms - independent search structure, set operations, and visible surface determination

- BSP tree -> all unified in a single structure
  - reduces the conceptual complexity and complexity of implementations
Assessment

Pros:

● Very detailed
● Not too complicated to follow
● Many diagrams to illustrate concepts
● Clear pseudocode

Cons:

● Could have provided more detail as to why approach is better
● Could have used better organization
Questions?