UCL

Binary Space Partition Trees

UCL

Overview

- Previous list priority algorithms fail in a number of cases, non of them is completely general
- BSP tree is a **general** solution, but with its own problems
  - Tree size
  - Tree accuracy

UCL

Binary Space Partitioning Trees
(Fuchs, Kedem and Naylor `80)

- More general, can deal with inseparable objects
- Automatic, uses partition planes defined by the scene polygons
- Method has two steps:
  - building of the tree independently of viewpoint
  - traversing the tree from a given viewpoint to get visibility ordering

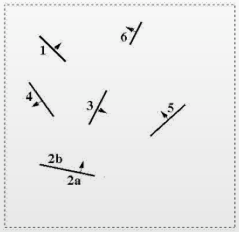
UCL

Binary Space Partitioning Trees

- BSP tree: organize all of space (hence *partition*) into a binary tree
  - *Preprocess*: overlay a binary tree on objects in the scene
  - *Runtime*: correctly traversing this tree enumerates objects from back to front
  - Idea: divide space recursively into half-spaces by choosing *splitting planes*
    - Splitting planes can be arbitrarily oriented

UCL

Building a BSP Tree (Recursive)

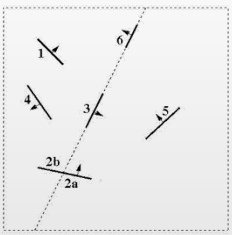


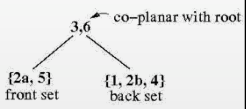
The tree is empty at first

A set of polygons {1, 2, 3, 4, 5, 6}

UCL

Building a BSP Tree (Recursive)

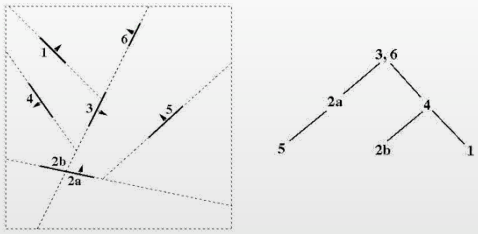




Select one polygon and partition the space and the polygons

UCL

Building a BSP Tree (Recursive)



Recursively partition each sub-tree until all polygons are used up

UCL

Building a BSP Tree (Recursive)

- Start with a set of polygons and an empty tree
- Select one of them and make it the root of the tree
- Use its plane to divide the rest of the polygons in 3 sets:
  - Front, back, coplanar
  - Any polygon crossing the plane is split
- Repeat the process recursively
  - with the front and back sets
  - creating the front and back subtrees respectively

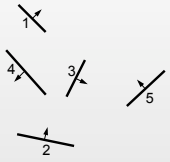
UCL

Building a BSP Tree (Incremental)

- Start with a set of polygons and an empty tree
- Insert the polygons into the tree one at a time
  - Insertion is done by comparing it against the plane at each node, and
  - propagating it to the correct side, splitting if necessary
- When the polygon reaches an empty cell, make a node with its supporting plane
- Results depends on insertion order

UCL

Building a BSP Tree (Incremental)

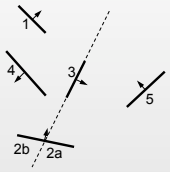


- Randomly start with a polygon, e.g., #3

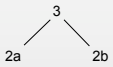
3

UCL

Building a BSP Tree (Incremental)



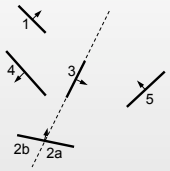
- Randomly, select next polygon, e.g., #2, insert into tree



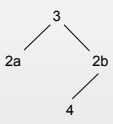
(left: front, right: back)

UCL

Building a BSP Tree (Incremental)



- Next one: #4

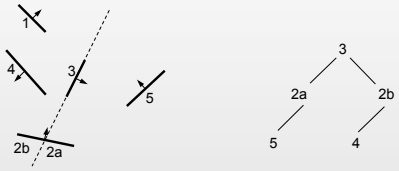


(left: front, right: back)

UCL

Building a BSP Tree (Incremental)

- Next one: #5

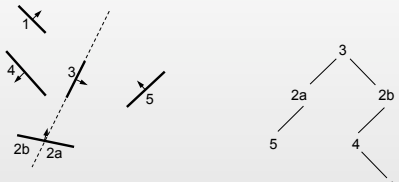


(left: front, right: back)

UCL

Building a BSP Tree (Incremental)

- Last one: #1



(left: front, right: back)

UCL

BSP Tree Traversal

- Why is a BSP tree useful at all?
  - Enumerate all polygons back-to-front for given viewpoint
  - (Use to accelerate ray-tracing)
- Java demo at:
  - http://www.symbolcraft.com/graphics/bsp/

UCL

BSP Tree Traversal

- How do we traverse the tree back to front for a given viewpoint?
  - Viewpoint given as position
- Recursive traversal:
  - If viewpoint is in front of plane (node)
    - Traverse its back first, then front
  - Otherwise
    - Traverse its front first, then back

UCL

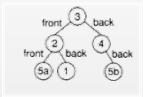
Back-to-Front Traversal

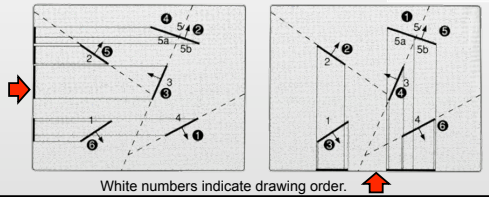
```
void traverse_btf(Tree *t, Point vp)
{
      if (t == NULL) return;
      endif

      if (vp in-front of plane at root of t)
            traverse_btf(t->back, vp);
            draw polygon of node t;
            traverse_btf;(t->front, vp);
      else
            traverse_btf(t->front, vp);
            draw polygon of node t;
            traverse_btf(t->back, vp);
      endif
}
```

UCL

Back-To-Front Traversal for Given Direction

- Camera point as dynamic classification point
- Traversal order:
  - parts behind the plane (w.r.t. direction)
  - polygon of the plane
  - parts in front of the plane (w.r.t. direction)

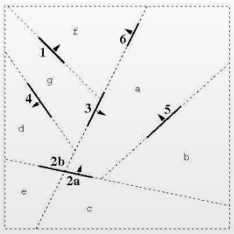


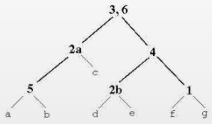


White numbers indicate drawing order.

UCL

BSP as a Hierarchy of Spaces

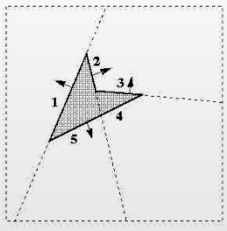


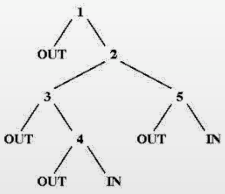


- Each node corresponds to a region of space
  - the root is the whole of R^n
  - the leaves are homogeneous regions

UCL

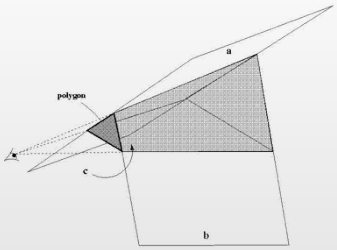
Representation of Polygons

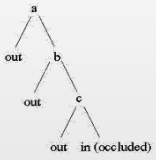




UCL

Representation of Polyhedra





UCL

BSP Trees for Dynamic Scenes

- When an object moves the planes that represent it must be removed and re-inserted
- Some systems only insert static geometry into the BSP tree
- Otherwise must deal with merging and fixing the BSP cells (see the book!)

UCL

Recap

- A BSP is a sequence of binary partitions of space
- Can be built recursively or incrementally
- Choice of plane used to split is critical
- BSP trees are hard to maintain for dynamic scenes