Visibility Algorithms II: Partitioning Trees

How to determine which surfaces are visible to the viewer and which are hidden and by how much

Input: Collection of polygons

Output: View Independent spatial data structure (partitioning trees) so that a visibility map can be obtained extremely fast for different viewpoints

Binary Space Partitioning Trees

A BSP (Binary Space Partitioning) tree is a recursive sub-division of space that treats each polygon, in 3D as a partitioning half-space and uses it to classify all remaining polygons in either of the two (binary) half-spaces. The half-space containing the normal can be considered the "front" half-space and the other the "back" half-space. In other words, when a partitioning half-space is inserted into the tree, it is first classified with respect to the root node, and then recursively with respect to each appropriate child.

Operations

- visibility orderings: viewer or light source dependent
- intersections: between geometric sets

Converting B-REPS to Partitioning Trees

Insert Figure

```
Brep_to_Bspt:  Brep b -> Bspt T
   IF  b == NULL
   THEN
      T = if a left-leaf then an in-cell else an out-cell
   ELSE
      H = Choose_HalfSpace (b)
      {b+, b-, b0}  = Partition_Brep (b, h)
      T.faces = b0
      T.pos_subtree = Brep_to_Bspt (b+)
      T.neg_subtree = Brep_to_Bspt (b-)
  END
```

Rendering Partitioning Trees: Generalized Painter's Algorithm

A partitioned scene is rendered by locating the eyepoint with respect to the root half-space, recursively rendering all polygons on the ”other” side, rendering the root polygon, and then recursively rendering the polygons on the same side of the eyepoint. Because each polygon is visited exactly once while drawing the scene, the scene can be rendered correctly in O(n) time.

```
render3DScene()
  if location(eye.point) == frontSide
    back.render3DScene()
    renderPolygon()
    front.render3DScene()
  else if location(eye.point) == backSide
    front.render3DScene()
    renderPolygon()
    back.render3DScene()
```

Intersections

Insert Figure

```
Merge_Bspts: (T1,T2:Bspt) -> Bspt

  Types
    BinaryPartitioner: {half-space, sub-half-space)
    PartitionedBspt: (inNegHs, inPosHs : Bspt)

  Imports
    Merge_Tree_with_Cell: (T1, T2 : Bspt) Bspt
    Partition_Bspt: (Bspt, BinaryPartitioner) PartitionedBspt

 Definition
  IF T1.is_a_cell OR T2.is_a_cell
  THEN
     Val:= Merge_tree_with_cell (T1,T2)
  ELSE
```

```
      Partition_Bspt (T2,T1.binary_partitioner) -> T2_partitioned
                    Val.neg_substrate :=
                                Merge_Bspts (T1.neg_subtree,T2_partitioned.inNegHs
                   Val.pos_substrate :=
                                Merge_Bspts (T1.pos_subtree,T2_partitioned.inPosHs
   END

   RETURN   Val
END  Merge_Bspts
```

Comparisons to z-buffer

- no numerical problems created by perspective projection
- no z-buffer memory
- unlimited use of transparency
- anti-aliasing without subpixel color and z-buffers
- no quantization errors for shadow computation which are amplified by the inverse perspective projection

Reading Assignment and News

Pages 541 - 543, of Recommended Text.

Also try BSP applet http://symbolcraft.com/graphics/bsp/index.html

Please also track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics25/cs354/)