Hidden Line/Surface Determination

• AKA visible line/surface determination

Eye/Camera/Object Space Algorithms
- Roberts Volume Intersection
- Schumacker’s List Priority Ordering
- BSP
- Painter’s
- Depth Buffer
- Scannline

Image Space Algorithms
- Mixed
- Area Subdivision
- Warnock’s

Object Versus Image Space

• Object Space
  - Done in eye/camera coordinates
  - Remove, then project
  - Geometric relationships maintained → zoom
  - Uses as much precision as possible
  - Efficiency is ∝ n², n is number of objects in viewing volume

• Image Space
  - Done in image space coordinates (normalized coords.)
  - Project then remove
  - More than one object may be mapped to same point on device
  - Efficiency ∝ f(complexity of visible objects)
    - Usually less complexity than in total scene

Common Elements in Techniques

• Sorting of objects
  - Facilitate depth comparisons

• Coherence Methods
  - Using regularities in scene

Sorting Issues

Algorithms

Eye/Camera/Object Space Algorithms
- Roberts Volume Intersection
- Schumacker’s List Priority Ordering
- BSP
- Painter’s
- Depth Buffer
- Scannline

Image Space Algorithms
- Mixed
- Area Subdivision
- Warnock’s
Robert's Volume Intersection

- First algorithmic solution to problem
- Within view volume, eliminate edges or planes from each volume that are hidden by volume itself
- Each remaining unobscured portion of edge is compared to each of remaining volumes to determine which portion(s) are hidden by these volumes.
- $O(n^2)$, $n$ is number of objects within view volume
- Can be improved by sorting and minmax test

Using A Shadow Box

Algorithms

Scanline Algorithm

- Requires two arrays the size of one line of the screen
  - Depth – initialized as ‘far away’
  - Color – initialized to background color
- Almost identical to depth buffer algorithm

Depth (Z) Buffer Algorithm

- Requires two arrays the size of the screen
  - Depth – initialized as ‘far away’
  - Color – initialized to background color
- Used in OpenGL

For each polygon,
  For each $y$,
    For each $x$,
      If $z < \text{depth}[x][y]$,
        $\text{Depth}[x] = z$
        $\text{Color}[x][y] = \text{polygon color}$
Warnock's Algorithm

- Hypothesis:
  - Sample areas called “windows” can be declared homogeneous
  - Can be displayed using a simple shade algorithm
- Uses coherency – not good on raster devices
- Approach: subdivide display into “windows”
  - Initially, one covering the entire display
  - Subdivides into ½ and ¼ size as needed
- Hypothesis is correct if
  - No faces fall within sample “window” at all
  - One face completely covers the “window” and is CLOSER TO the viewpoint than every other face

Warnock's Algorithm

- Procedure for testing is called a cull
  - Set of faces is compared to window for checking
  - Test for window need not test all faces
    - E.g., those which have been eliminated with bigger window
  - Depends only upon x-y coordinates, not z
- Several possible situations:
  - Face surrounds sample window
  - Face overlaps sample window
  - Face disjoint from sample window

Painters Algorithm

- Just uses color array
- First sort polygons furthest to nearest
- Render polygons in sorted order
**Schumacker’s List Priority Ordering**

- Based on *cluster* concept
  - Collections of faces
  - Clusters on the same side of a plane as the eyepoint can obscure those on the other side

**Schumacker’s List Priority Algorithm**

- **Object Space**
  - Depth overlap calculations are performed with high precision
  - Analysis ahead of time stored in DB
  - Scene broken into clusters, each of which has high priority face that obscures other faces

- **Image Space**
  - Viewpoint entered and used to search for cluster
  - If two points project to same pixel, the one with highest priority is displayed
  - Real-time processing

**Algorithms**

- Roberts Volume Intersection
- Schumacker’s List Priority Ordering
- Area Subdivision
- Scanline
- Painters
- BSP
- Warnock’s

**BSP Tree Creation – View Independent**

- Based on Schumacker’s work
- Choose a face to be at the root
- Split other faces into two groups and place them as follows:
  - If the face is on the FRONT of the root node, place it in the left list
  - If the face is on the BACK of the root node, place it in the right list
- Recursively process the front list
- Recursively process the back list

**BSP Tree Example**

**BSP Traversal – View Dependent**

- Applied recursively at each node
- To draw the faces stored in the BSP tree with root F:

  ```plaintext
  if( eye is in the "back" half-space of face F ) {
      draw all faces in the front subtree of F;
      draw F;
      draw all faces in the back subtree of F;
  } else { // eye on "front" of face F
      draw all faces in the back subtree of F;
      draw F (if you want to draw back faces);
      draw all faces in the front subtree of F;
  }
  ```