Hidden Surface and Hidden Line Removal

Task: Identification and Removal of parts of picture that are not visible from a chosen viewing position.

Classification of Algorithms:
1. Object-Space Algorithms:
   algorithms that deal with the object definition directly
   visibility is decided by Line of surface by line or surface
   • Back-Face Removal
   • Depth Sorting (Painter Algorithm)
   • Area Subdivision (Quadtrees and Octrees)
   • Binary Space Partition Trees

2. Image-Space Methods
   algorithms that deal with the projected image
   visibility is decided point by point
   • Depth Buffer (Z-buffer)
   • Scan-line
   • Depth sorting (Painter Algorithm)

Both classes use sorting and coherence methods:
sorting – to facilitate depth comparison
coherence – to exploit regularities in a scene
**Hidden Surface Removal**

- **Viewing**
- Looking along any projector (from center of projection, for example) we see one or more surfaces.

**Hidden Line**
- when outline of an object is to be displayed
- similar to clipping a line segment against a window
- most surface algorithms can be applied for hidden line elimination.

**Hidden Line Removal**
- assumes that objects are modeled with lines.

- lines are created where surfaces meet

- if only the visible surfaces are drawn, the invisible lines are automatically eliminated.
ABCD, DCEF, ADFG are visible. Therefore, all lines are visible, that is, AB, DC, FE, AG, … , BC, CE, GF.

AGHB, GHEF, BHEC are invisible surfaces. Therefore, BH, HE, … are invisible and must be removed.

Hidden Surface Algorithms
-Clipping

-Image Space algorithms
  • Z-Buffering
  • Painter’s Algorithm

- Object Space Algorithms
  • Binary Space Partition Trees
  • Back-Face Removal

Example:

if we know that A is in front of B, and B is in front of C, we can fill or paint C followed by paint B, and finally paint A onto the display.
The Concept:
- if we have \( K \) objects,
  take one of the objects and determine its relationship to each of the other \( K-1 \) objects, then:
either : paint
discard
partially paint
- repeat recursively:
  we do this operation:
  \((K-1)+(K-2)+...+2+1 = K(K-1)/2\) times

Depth Buffer Method
- z-buffers are needed
  (1) depth buffer to store \( z \)-values for each \((x,y)\) positions as surfaces are compared
  (2) refresh buffer to store the intensity values (colours) for each position

- depth is calculated from the plane equation : \( Ax + By + Cz + D = 0 \)
  \(\Rightarrow z = \frac{-Ax - By - D}{C}\)

* For scanlines, \( x \)-coordinate differ by 1 across the line and \( y \)-coordinate differ by 1 between lines

* if for a position \((x,y)\) we determined \( z \), then the depth of the next position \((x+1, y)\), \( z' \)
  along a scanline is: \( z' = \frac{-A(x+1) - By - D}{C} = z - \frac{A}{C} \).
Binary space partition (BSP) Tree
(Fuchs, Kedem, and Naylor 1983)
(based on Schumacker’s work 1969)

- good for when viewpoint changes, but the objects do not.

- Schumacker:
  environments can be viewed as being composed of clusters.
  - if a plane can be found that wholly separates one set of clusters from another, then
  clusters that are on the same side of the plane as the eyepoint can be obscured, but
  cannot be obscured by clusters on the other side.
  - proceed recursively

- In simple terms
  - scan convert opposite clusters, then the root and finally scan convert the near side
  cluster.

Example: BSP-tree

- one plane
  - if viewpoint is in front of $P$
    - paint 2, then 1
    - otherwise, paint 1 and then 2

  - clusters are represented in binary tree with
    - plane(s) as a node(s)
    - clusters are the leaves
Example:

- All polygons from the other side of P from the viewpoint are painted first
- Paint P, the root,
- Paint polygon on side of eyepoint

**Depth sorting**
(painter’s algorithm)

a. surfaces are sorted in order of decreasing depth (in object space)

b. surfaces are scan-converted in order, starting with the surface of greatest depth (in image space)