Visibility Determination

Outline
• What is visibility determination problem
• Classification of different approaches
• Commonly used techniques:
  • Back-face culling
  • Painter's algorithm
  • Z-buffer algorithm
  • Scanline algorithm

Visibility Determination Problem
• Objective:
  • Provide a better 3D effect by display only visible surfaces or visible parts of surfaces
  • Also called:
    • Visible surface determination
    • Hidden surface removal/elimination

Classification of Different Approaches
• Conservative visibility algorithms:
  • Back-face culling
• Object space algorithms:
  • Painter's algorithm
• Image space algorithms:
  • Z-buffer algorithm
  • Scanline algorithm
  • Ray cast algorithm

Conservative Visibility Algorithms
• Simplify the visibility task without necessarily solving it completely
• Assumes the presence of other visibility algorithms to resolve exact visibility
  • Either an object space algorithm or an image space algorithm
• Normally used as a pre-processing technique

Back-Face Culling
• Polygons not facing the viewpoint are never visible
  • These polygons can be removed from the rendering process
• Assumptions:
  • The scene is composed entirely of opaque & closed surfaces
  • The normals of all polygons are consistently defined
Back-Face Culling (Cont’d)

- **Algorithm:**
  - for (each polygon P) {
    - Vector V = any point on P - viewpoint;
    - Vector N = normal of P;
    - if (V \cdot N \geq 0)
      - Ignore P in further rendering process;
  }

- **Effects:**
  - Remove about half of the polygons in the scene

- **Complexity:**
  - O(n), quite efficient

Object Space Algorithms

- **Compare objects with each other**
  - Eliminate invisible objects or invisible portion of an object

- **Works in world space**
  - Independent to the resolution of image to be generated
  - Visibility determination results can be reused as long as the viewpoint does not change

Painter’s Algorithm

- **Simulate how artists draw on canvas**

- **Sort polygons by depth**
  - Split some polygons if necessary

- **Draw each polygon from back to front**
  - Normally use scan conversion
  - Need to calculate the color even an object is invisible

Why Needs to Split?

- **Visibility cycle:**
  - Intersecting polygons:

Algorithm Evaluation

- **Advantages:**
  - No extra storage needed
  - Can handle transparent polygons

- **Disadvantages:**
  - Sorting is expensive
    - When viewpoint changes, sorting has to be redone
  - Some polygons have to be split before sorting
    - Splitting operation is not trivial
    - Increase the number of polygons for sorting

Image Space Algorithms

- **Assume the user has already specified the resolution of the image to be generated**
  - Determine which object is visible at each pixel on the image

- **Works in image space:**
  - Dependent on the resolution of image to be generated
  - Need to recalculate if the user needs to render the scene in a different resolution
Z-Buffer Algorithm
- Project each polygon to the image plane
- Check all the pixels within the projection
- If the current polygon is the closest so far at a given pixel:
  - Update the pixel’s color according to the current polygon

Z-Buffer Example

Z-Buffer Algorithm (Cont’d)
- for ( int y=0; y<height; y++ )
  - for ( int x=0; x<width; x++ ) {
    - pixel[x][y] = background;
    - zBuffer[x][y] = +∞;
  }
- for ( each polygon P ) {
  - project P to the raster coordinates;
  - for ( each pixel (x,y) in the projection ) {
    - float z = depth of P at coordinate (x,y);
    - if ( z < zBuffer[x][y] ) {
      - zBuffer[x][y] = z;
      - pixel[x][y] = P’s color at (x,y);
    }
  } }

Scanline Algorithm
- Step through each scanline in the image
- The scanline and the viewpoint define a plane
  - Only polygons that intersect with plane need to be considered
  - Problem becomes 2D:
    - Determine which intersection (line segment) is visible

Scanline Algorithm (Cont’d)
- Use a sorted array to keep the order of line segments
- Update order when:
  - A new line segment starts
  - An existing line segment ends
  - Two line segments intersect
  - The closest line segment is used to calculate color

Z-Buffer vs. Scanline
- Operate polygon by polygon
  - Need additional memory space for depth buffer
  - Do not support transparent objects
  - Colors for some invisible objects are calculated
  - Easy for parallel implementation on graphics hardware
- Operate scanline by scanline
  - Need memory space for polygon table, edge table, & active-edge list
  - Support transparent objects
  - Colors for invisible objects are not calculated