Lecture 06 - Visible-Surface Detection

I. Motivation

A. Which triangles do you draw? Yes?
   B. How do you get them drawn in the right order?

C. Visible-Surface Detection - determining what is visible in a scene from a chosen view position

1. Object-space approaches classify based on object definitions
2. Image-space approaches classify based on projected images

C. Visible-Surface Detection - determining what is visible in a scene from a chosen view position

D. We can use any/all methods in conjunction.

II. Back-Space Culling (Object Space Approach)

A. A back-space is an object (primitive) that is not oriented to be viewed by the camera.

   e.g. 
   
   Notice surface N is hidden by not shaded surface.

B. Consider the eye position of the camera (X, Y, Z). It is behind N if

   \[ A_1 + B_2 + C_3 + D < 0 \]

C. More simply we can look at the dot product of N and \( z \).

   A. Back face satisfies:

   \[ \hat{z} \cdot N > 0 \]

D. When in projection coordinates \( \hat{z} \) is Z-axis of the space and we only need to consider Z-component \( \hat{z} \cdot N \). A polygon is a back face when

   \[ C < 0 \]

   *What if \( C = 0 \)?

E. For convex polyhedron, works perfectly. All hidden surfaces removed.

   For concave polyhedron, only eliminates some, other surfaces might be hidden or partially hidden.

   However, back face culling can be expected to eliminate 1/2 of faces.

F. Problem: Discuss in parts. How could you draw inner shell of an object? Example:

   a boundary of a space, without blocking view from user.
III. Depth-Buffer (z-buffer) (image space approach)

A. Idea - compare surface depth values in a plane for each pixel. If a nearer surface is seen use pixel color.

B. Surfaces can be processed in any order. Depth-buffering usually uses in homogeneous coordinates.

C. A separate memory space, from the frame buffer (color), is needed. The depth buffer stores depth values initially (or cleared) to 1.0 (near depth).

D. Algorithm

1. Initialize depth buffer + frame buffer of all pixels (x,y) to depth (x,y) = 1.0. Some (x,y) = background color.

2. Process each polygon as follows:
   a. For each projected (x,y) calculate its depth z.
   b. If z < depth (x,y) compute new color at (x,y) and depth (x,y) = z. Some (x,y) = color.

E. Computing depth values

i. Initial idea: \( z = \frac{-B_0 - B_1 - B_2 - B_3}{C} \) from plane equation. How can we improve efficiency?*

   ii. However, we should exploit scan lines. X positions divide by \( \pm 1 \) as do y-values. So if depth of (x,y) is known as \( z \), \( z' \) of previous \((x+1,y)\) can be

   \( z' = \frac{-A(x+1) + B_1 - B_2 - B_3}{C} = \frac{-A - B_1 - B_2 - B_3}{C} = z - \frac{A}{C} \) *incremental update*

iii. Beginning depth of scan line can be computed from previous scan line as well:

   \( x' = x - \frac{A}{C} \) and \( z' = z + \frac{B_1 + B_2}{C} \) or \( z' = z + \frac{B_2}{C} \) for vertical line.

   Could also use a Bresenham approach here.

F. Discussion? pass. cons?

i. Gets objects drawn in correct order.

ii. Does not need expensive updates of frame buffer.

iii. Cannot handle transparencies.

G. Problem: Discuss in pairs: Say you want to render a complicated background and foreground objects that are transparent. And you want transparencies blended. How could you do this if some objects may lie behind background?
H. Extension - A Buffer (accumulation buffer)

- Handles anti-aliasing, visibility detection, transparency, etc.

- Each pixel has
  a. Depth (positive, negative, or zero)
  b. Surface data - as in depth buffer

- Idea
  a. If depth is non-negative, the surface is visible.
  b. If depth is negative, there are multi-surface contributions. So the series of surfaces stored in pixel are "accumulated" to get final color.
  c. Can be implemented w/ scan lines.

IV. Scan-line Method (image space)

A. Scan - compute and compare depths across scan lines. Determine visible surface at each pixel and enter color into frame buffer.

B. Recall poly: S11 w/ edge table. Active edges stored update information for edge, s1, s2, s3

C. We track active edges on a scan line sorted by increasing x-value. We also have a flag for each surface as "on" or "off".

D. Processing left to right - at a left intersection a surface is turned on, and at a right intersection it is turned off. Ask: how this saves computation vs. depth buffer.

E. Example:

![Diagram](image)

F. Take advantage of coherence for incremental updates and reduction of depth computations.

G. Discussion

- Can handle number of surfaces
- How about the following? No, why? *Ask class then to solve*

![Diagrams](images)
V. Depth-Sorting Method (image and object space)

A. Basic Idea

1. Sort surfaces in terms of decreasing depth

II. Scan-convert surfaces in order starting w/ greatest depth

This is often referred to as the painter's algorithm. A painter starts w/ background and then adds closer and closer objects. Avoids need for depth buffer.

6. Process

7. Order defined as smallest z-value on surface

8. In order, farthest surface is scan-converted after second- farthest surface

For depth overlaps, if no overlap occurs, greatest is scan-converted. This means that all are scan-converted, but no overlap in depth.

9. Additional tests - if one is true, no reordering is necessary

a. The bounding rectangles in xy directions do not overlap
b. Surface S is completely behind S' in relation to view (performed w/ back-front polygon test)

c. Overlapping surface is completely in front of S relative to view

d. Boundary-edge projections of the two surfaces do not overlap

e. If all fail, swap S, S' in sorting. Ex.

S. Retest S' if swapped

C. Discussion

10. Infinite loop possible - add flags on surfaces
11. Cyclic overlap - divide in parts.

VI. Binary Space-Partitioning Tree (BSP-Tree) Method (object-space)

A. Efficient technique for painter's algorithm when reference point changes

B. Construction

i. Subdivide scene into two sections at each step w/ a plane, P

ii. Repeat until each object is in each leaf

iii. Example
C. Drawing:
   i. Relative to viewpoint, begin with root node perform a depth-first traversal
      ii. If viewpoint is in front of plane, draw back subtree first and then the front.
      iii. If viewpoint is behind plane, draw front subtree first and then the back.

VII. Area Subdivision Method (image space)
   1. Analyzes areas of objects in scene. Recursively subdivides areas of the view plane
      until each rectangle contains a single surface, no surfaces, or is pixel inside.
   2. Need tests to identify areas as single surface etc. Details in book.
   3. Example (similar to quadtree)

VIII. Octree Method (object space)
   1. Create oct-tree subdividing scene and use it to order objects.
      - Each node can have 8 children
      - Generalization of quadtree
      - Similar to BSP tree but with orthogonal planes

IX. Foreground is in boxes 9, 2, 3 (closest), background: 4, 5, 6, 7 (furthest)
C. Process in order 9, 3, 1, 4, 5, 7 (depth-first traversal). When color is determined
   it is saved in front quadtree (viewplane). Any node that is obscured does not
   need to be processed. Example: Grass is colored then back object does not need
   to be processed.

0. can receptor octree used on viewpoint.

IX. Ray-casting method (camera + object space)
   A. For each pixel, cast ray into space computing intersections of surfaces. Closest surface
      determines color.

B. Similar to ray-tracing (discussed later) and depth buffer.
II. Depth-Cueing (Image space)
   a. show distant objects less bright
   b. Eq.
      \[ f_{\text{depth}}(d) = \frac{d_{\text{max}} - d}{d_{\text{max}} - d_{\text{min}}} \]
      \[ \text{Color}' = \text{Color} \times f_{\text{depth}}(d) \]
   c. Similar to Sun, or other atmospheric effects.