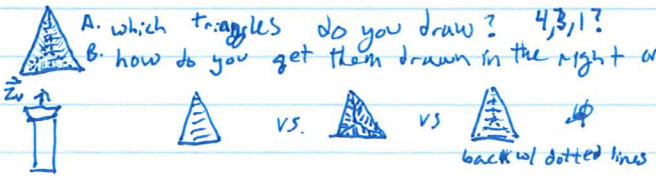
Lecture 06 - Visible-Surface Detection

I. Motivation -



A. which triangles do you draw? 4, 3, 1?
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

i. Object-space approaches classify based on object definitions

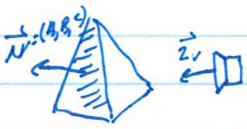
ii. Image-space approaches classify based on projected images

D. Can use any/all methods in conjunction

II. Back-Face Culling (object space approach)

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

e.g.



Notice surface N is hidden by non shaded surface

B. consider the eye position of the camera (x, y, z). It is behind surface (A, B, C, D) when

$$Ax + By + Cz + D < 0$$

C. More simply we can look at the dot product of N and $\vec{z}_v$

A back face satisfies:

$$\vec{z}_v \cdot N > 0$$

D. When in projection coordinates $\vec{z}_v$ is z-axis of the space and we only need to consider z-component, C, of N. A polygon is a back face when

$$C < 0$$

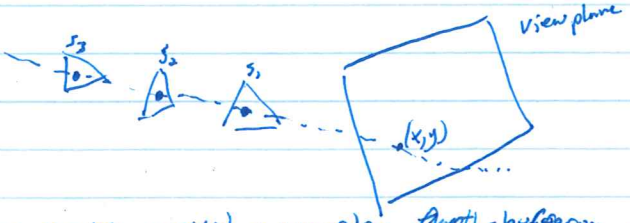
what if $C = 0$?

E. For convex polyhedron, works perfectly. All hidden surfaces removed. *what about concave?* 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 pairs: 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 scene for each pixel. If a nearer surface is seen update pixel color. (distance from view plane)



B. Surfaces can be processed in any order, depth-buffering usually occurs in homogeneous coordinates

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

D. Algorithm

1. Initialize depth buffer + frame buffer of all pixels (x,y) to
   depth (x,y) = 1.0 Frame (x,y) = background color
2. Process each polygon as follows
   - for each projected (x,y) calculate the depth z.
   - If z < depth (x,y) compute color at (x,y) and
     depth (x,y) = z Frame (x,y) = color.

E. Computing depth values

i. Initial idea: $z = \frac{-Ax - By - D}{C}$ from plane equation *How can we improve efficiency?*

ii. However, we should exploit scan lines. x positions change by ±1 as do y-values. So if depth of (x,y) is known as z, z' of position (x+1, y) can be

$$z' = \frac{-A(x+1) - By - D}{C} = \frac{-Ax - By - D}{C} - \frac{A}{C} = z - \frac{A}{C}$$ *incremental update*

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

$x' = x - \frac{1}{m}$ and $z' = z + \frac{A/m + B}{C}$ or $z' = z + \frac{B}{C}$ for vertical line

could also use a Bresenham approach here.

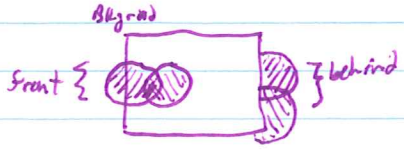
F. Discussion? pros? cons? *Ask class*

i. Gets objects drawn in correct order

ii. Does needless updates of frame/depth buffer.

iii. Cannot handle transparencies.

G. *problem: Discuss in pairs: Say you want to render a complicated background w/ foreground objects that are transparent. And you want transparencies blended. How could you do this if some objects may be behind background?



H. Extension - A-buffer (accumulation buffer)

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

ii. Each pixel has

a. Depth - (positive, negative, or zero)

b. Surface data - as linked list

iii. Idea

a. If depth is non negative, the surface is singular

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)

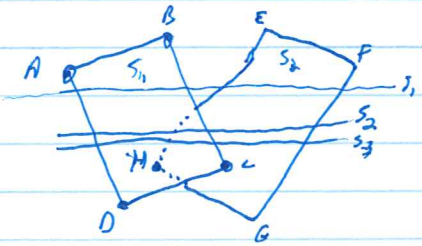
A. Idea - compute and compare depths across scan lines. Determine visible surface at given pixel and enter color into frame buffer.

B. Recall polygon fill w/ edge table. Active edges stored update information for edge, eg. slope

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:



S_1: AB, BC, EH, FG

S_1 on. S_2 on

S_2, S_3: AD, EH, BC, FG

S_1 S_2

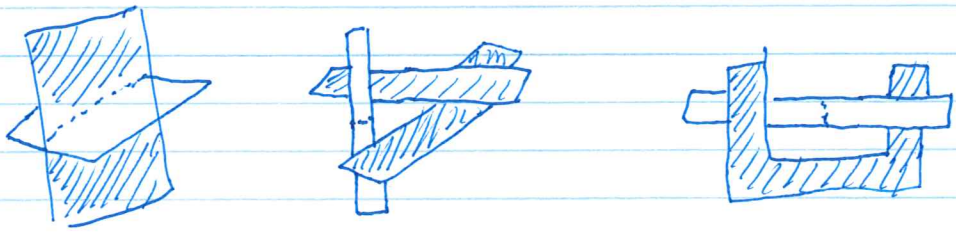
can compute depth value once at EH intersection to know visible surface

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

G. Discussion

i. Can handle and number of surfaces

ii. How about the following? No. Why? *Ask class them to solve?*



iii. Can subdivide surfaces, eg., on dotted lines

V. Depth-sorting Method (image and/or object space)

A. Basic Idea

i. Sort surfaces in terms of ~~increasing~~ decreasing depth

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

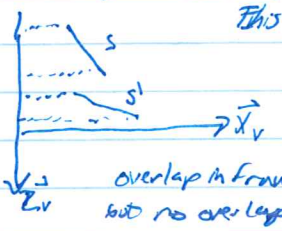
iii. often referred to as the painter's algorithm. A painter starts w/ background and then adds closer and closer objects. Avoids need for depth-buffer.

B. ~~Sorting~~ Process

i. Sorting: Order defined on smallest z-value on surface

ii. In order furthest surface is compared w/ second furthest surface for depth overlaps. If no overlap occurs greatest is scan converted. This repeats until all are scan converted.

If depth overlap occurs additional tests required.



overlap in frame buffer but no overlap in depth.

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

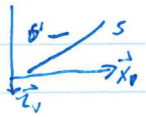
a. The bounding rectangles in xy directions do not overlap

b. Surface S is completely behind S' in relation to view (performed w/ back-front polygon tests)

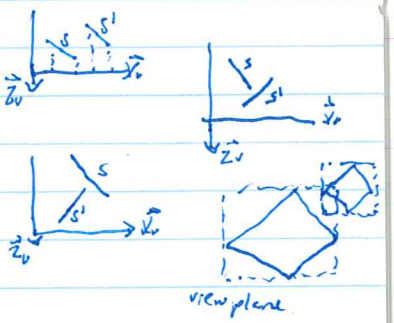
c. Overlapping surface is completely in front of S relative to view

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

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



f. Retest S' if swapped



C. Discussion

i. Infinite loop possible - add flags on surfaces

ii. Cyclic overlap - divide in parts.

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

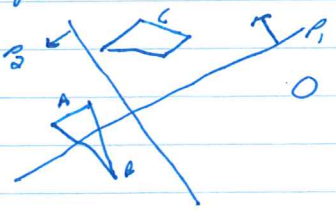
A. Efficient technique for painters algorithm when reference point changes

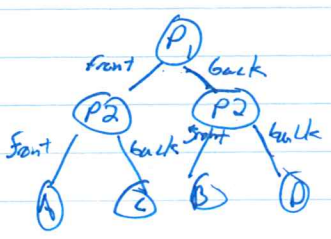
B. Construction

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

ii. Repeat until single object is in each leaf

iii. Example





C. Drawing -

i. Relative to view position, beginning w/ root node perform a ~~post order~~ traversal

ii. If viewpoint is in front of plane, draw back subtree first and then the front.

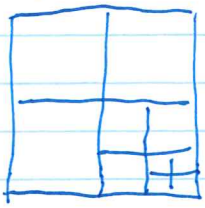
iii. If viewpoint is behind plane, draw front subtree first and then the back.

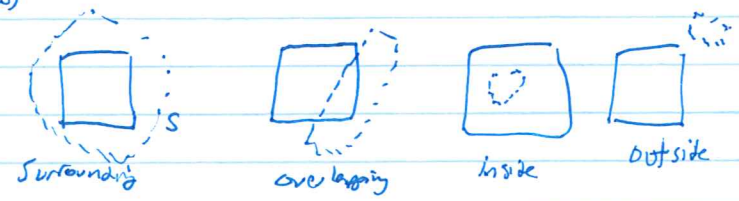
VII. Area-Subdivision Method (image space)

A. Analyzes area of objects on scene. Recursively subdivides areas of the view plane until each rectangle contains a single surface, no surfaces, or is 1 pixel in size.

B. Need tests to identify area as single surface etc. Details in book

C. Example (similar to quadtree)

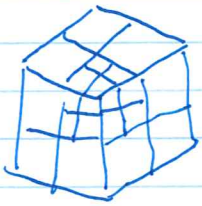




D. Subdivision leads to closest surface + color for each partition

VIII. ~~Octree~~ Oct tree Method (object space)

A. Create oct-tree subdivisions of a scene and use it to order objects



- Each node can have 8 children
- Generalization of quadtree
- Similar to BSP tree but w/ orthogonal planes

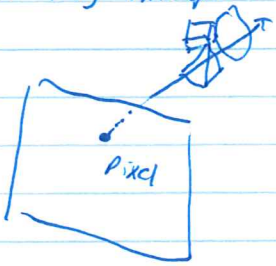
B. Foreground is in boxes 0,1,2,3 (closest), background in 4,5,6,7 (farthest)

C. Process in order 0,1,2,3,4,5,6,7 (depth first traversal), when color is determined it is saved in front quad tree (view plane). Any node that is obscured does not need to be processed, example front object is colored then back object does not need to be processed.

D. Can reorient ~~Octree~~ Oct tree based on viewpoint.

IX. Ray casting method (image + objects 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~~ depth-buffer.

IX. Depth-Cueing (image space)

A. Show distant objects less bright

B. Eq.

$$f_{depth}(d) = \frac{d_{max} - d}{d_{max} - d_{min}}$$

$$Color' = Color * f_{depth}(d)$$

C. Similar to fog. or other atmospheric effects.