UCL

Visibility Determination

UCL

Overview

- Why visibility culling?
  - Avoid *incorrect* image being displayed
  - Enables speed-up
- With ray casting, visibility was solved implicitly in image space
  - Now we are projecting polygons, we need to address this directly
    - Z-Buffering is one solution to it
    - We want to look at other solutions

UCL

Types of Visibility Methods

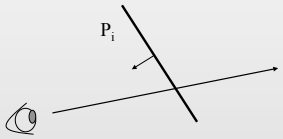
- Back-face elimination
- List priority
  - Ordering in Projection Space
  - Depth-sort
  - Ordering in Object Space
  - Binary Space Partition Trees
- Image precision, e.g.
  - Z-buffer
  - Ray Casting

} Already seen these methods

UCL

Back Face Culling

- We were specifying the order (clockwise or counter clockwise) for a reason
- Polygons whose normal does not face the viewpoint, are not rendered



UCL

Back Face Culling

- Thus if we have the plane equation

$$l(x, y, z) = ax + by + cz - d = 0$$

- and the COP (cx,cy,cz), then

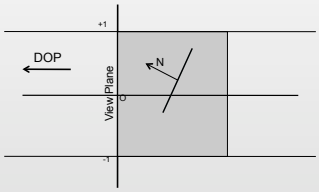
$$l(c_x, c_y, c_z) > 0$$

- if the COP is in *front* of the polygon
- Otherwise: cull

UCL

Back Facing Culling

- Alternatively, think about projection space
  - The transformed polygon must have a normal N with a negative z component to be visible



UCL

Calculating the Normal (of transformed poly)

- If the plane equation is expressed so:

$$p \cdot l = 0$$

$$(x, y, z, 1) \cdot \begin{bmatrix} a \\ b \\ c \\ -d \end{bmatrix} = 0$$

Reminder: (a, b, c) is the normal of the plane

- Suppose T is the modelview matrix and thus transformed point q is

  q = pT

UCL

Calculating the Normal (of transformed poly)

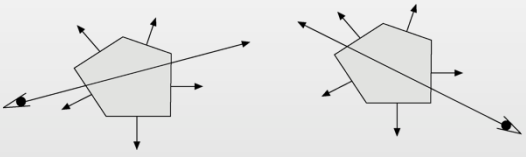
- If q = pT
- Then $qT^{-1} = p$
- So $qT^{-1}.l = p.l$
- Let $m = T^{-1}.l = l.(T^{-1})^T$
- Then q.m = 0 (= p.l)

- Therefore transform plane equation by inverse transpose of modelview transform
- Note, that x-formed normal might need rescaling

UCL

Back Face Culling

- Final words – for a *single convex* object, ANY order of polygons is correct (assuming backface culling)



UCL

Back Face Culling Summary

- Enables **speedup** by not drawing unnecessary polygons
- Does not ensure correct drawing order for visible surfaces, unless there is only a single convex object

UCL

Visibility Methods

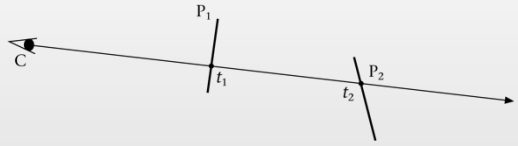
- Back-face elimination
- List priority
  - Ordering in Projection Space
  - Depth-sort
  - Ordering in Object Space
  - Binary Space Partition Trees
- Image precision, e.g.
  - Z-buffer
  - Ray Casting

} Already seen these methods

UCL

Visibility (Priority) Ordering – Painters Algorithm

- Given a set of polygons *S* and a viewpoint *C*: Find an ordering on *S*
  - such that for any 2 polygons intersected by a ray through C, P_i has higher priority than P_j



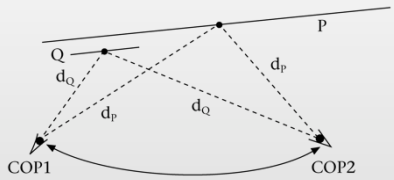
- You would render P_2 THEN P_1 to see correct result

Painters Algorithm

UCL

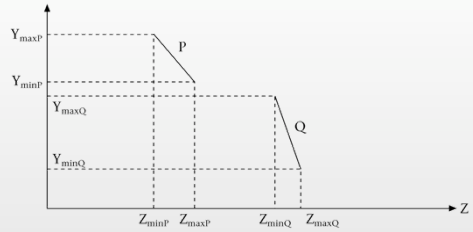
Z-Sort in Projection Space

- Simply sort back to front all polygons based on their mid-point
- Doesn't always work, e.g.



UCL

Depth-Sort (Newell et al. 72)



- Note that two polygons P&Q can be rendered in any order, if they do not overlap in X-Y
- P can be rendered before Q if P's z range is completely behind Q's

UCL

Depth-Sort

- P can be rendered before Q if
  - Z-extent of Q is wholly in front of P **or**
  - Y-extent of Q does not overlap P **or**
  - X-extent of Q does not overlap P **or**
  - All points on P lie on the opposite side of Q than the COP **or**
  - All points on Q lie on the same side of P as the COP **or**
  - The projections of P and Q on the XY plane do not overlap (full 2D polygon overlap test)
- Conclusions: expensive

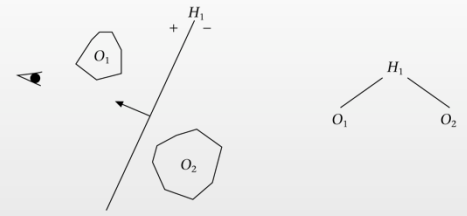
UCL

Object Space Methods

- Depth-sort tells us the relationship between a set of polygons, but must be computed every time
- However we know that most of the time objects do not intersect
- Can we work in object space to find gross relationship between the objects O1 and O2 so that we know all polygons in O1 can be rendered before O2 ?

UCL

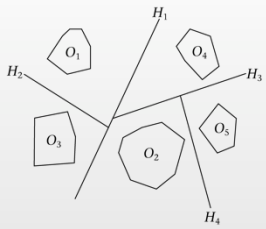
Schumacker 69

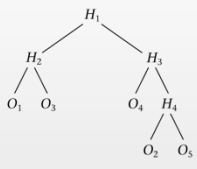


- You can define a plane such that: a polygon/object on the same side as the viewpoint has higher priority than one on the opposite side

UCL

Schumacker 69





- If we have more than one object on one side then repeat the same reasoning and add more partitioning planes between these objects

UCL

Other Uses of Culling

- Done enough to ensure ***correct*** view, but we can do more to ensure ***speed*** (in the advanced course, next semester)
- View Volume Culling
- Visibility Culling
- Occlusion Culling

UCL

Summary

- Distinction between types of visibility algorithm
  - Projection space
  - Object space
  - Image space
- Back face culling as a *preliminary* step
- How priority ordering can be achieved through depth-sort and object-space methods