
CS770/870 Fall 2008

Visible Surface Algorithms

Related material in Hill and Kelley: 8?

10:05

CS770/870 Fall 2008 Bergeron/Plumlee

1

Preview

- Image space algorithms
- Object space algorithms

10:05

CS770/870 Fall 2008 Bergeron/Plumlee

2

Evaluation Issues

- There are a few key issues to consider regarding all visible surface algorithms
 - Is algorithm general purpose?
 - What is the space/time performance?
 - How easy is it to incorporate improved realism?
 - Is it appropriate for the particular task

10:05

CS770/870 Fall 2008 Bergeron/Plumlee

3

Image Precision v. Object Precision

- Image precision algorithms
 - Work in *image space* (in floating point)
 - for each pixel
 - find closest object that projects onto pixel
 - color the pixel with color of object
 - Computations done at the precision of the image
- Object precision algorithms
 - for each object in the world
 - find the parts not obscured by other objects
 - draw these parts where they belong on display
 - Computations done at the precision of the object space
 - have to deal with a pixel partially covered by the object

10:05

CS770/870 Fall 2008 Bergeron/Plumlee

4

Coherence

- Different algorithms take advantage of 1 or more forms of *coherence* to improve efficiency
 - *object coherence*: objects tend to be compact and separated from other objects
 - *face coherence*: surface properties vary smoothly across object faces
 - *edge coherence*: edge visibility can only change if the edge crosses a visible edge or intersects a visible face
 - *implied edge coherence*: the intersection of two planar is defined by a simple edge with 2 end points
 - *scanline coherence*: neighboring scanlines have similar visibility

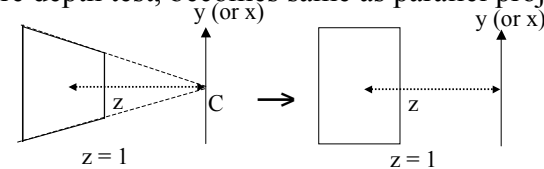
10:05

CS770/870 Fall 2008 Bergeron/Plumlee

5

Comparing Depths

- Determining visibility relies on comparing the “depths” of 2 objects with respect to the display
 - Transform world so display screen located in xy plane
 - z-values can then be used to determine closeness to display
 - *parallel* projections: all points that project to pixel (x,y), have same x,y values
 - *perspective* projections: if apply *perspective transformation* before depth test, becomes same as parallel projection



10:05

CS770/870 Fall 2008 Bergeron/Plumlee

6

Bounding Areas and Volumes

- Lots of optimizations can be achieved using *bounding boxes* in either 2D screen or 3D world. E.g.,
 - If bounding boxes of the projections of 2 objects don't overlap in the images, the objects cannot obscure each other
 - If bounding boxes of objects in 3D don't overlap, can simplify many calculations
 - *minmax* tests compare bounds of 1 dimension of the box
- Other bounding volumes
 - spheres
 - cylinders
 - parallel planes

10:05

CS770/870 Fall 2008 Bergeron/Plumlee

7

Self-Obscuring Tests

- Objects in the scene are composed of faces that are obscured by other (opaque) faces
 - on average half the faces are obscured by other faces
 - these are called backfaces
- Compute the normal to the polygon that points outside the volume, the outward normal, N
- Create a vector from the polygon to the eye point, E
- If the angle between N and E is $> 90^\circ$, the viewer is looking at the “back” of the face, so it is invisible
- This is called backface culling

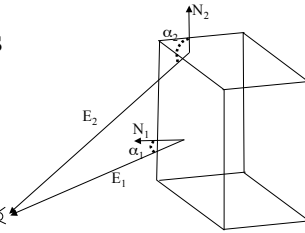
10:05

CS770/870 Fall 2008 Bergeron/Plumlee

8

Backface Culling

- Compute the outward normal
 - Ordering of vertices determines direction of normal
- Create vector from any point on polygon to eye
- A back face normal makes an angle $> 90^\circ$ with eye vector
- The computation can be done either in world coordinates or viewing coordinates



$$\alpha_1 = E_1 \cdot N_1 < 90^\circ$$

$$\alpha_2 = E_2 \cdot N_2 > 90^\circ \text{ backface}$$

Backface Culling Details

- Don't need to compute the angle, just the cosine
 - $\alpha > 90^\circ$ iff $\cos \alpha < 0$
- Don't need to compute cosine either
 - $E \cdot N = \|E\| \|N\| \cos \alpha$
 - So, $\cos \alpha < 0$ iff $E \cdot N < 0$
- Can apply the same logic to a parallel projection
 - the eye is at ∞ along the direction of projection
 - so face is a backface if $\mathbf{dop} \cdot \mathbf{N} < 0$.
 - if $\mathbf{dop} = (0 \ 0 \ 1)$, $\mathbf{dop} \cdot \mathbf{N} = 0 \cdot N_x + 0 \cdot N_y + 1 \cdot N_z = N_z$
 - also true if apply test after perspective transformation

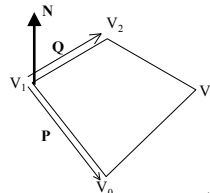
Plane Normal from Planar Vertices

- Compute cross product of any 3 non-collinear vertices:
 - Pick any vertex, define vectors from that vertex to neighbors
 - Let $P = (V_0 - V_1)$ and $Q = (V_2 - V_1)$, then

$$\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix} - \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix} - \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix} - \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ P_x & P_y & P_z \\ Q_x & Q_y & Q_z \end{vmatrix}$$

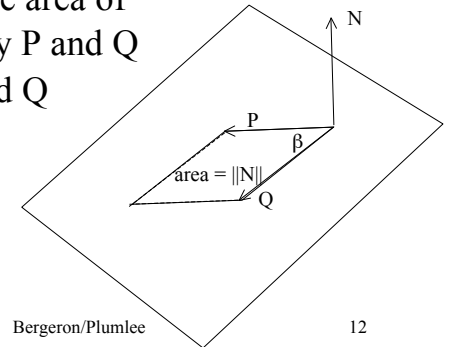
$$= \mathbf{i}(P_y Q_z - P_z Q_y) - \mathbf{j}(P_x Q_z - P_z Q_x) + \mathbf{k}(P_x Q_y - P_y Q_x)$$

- \mathbf{N} is normal to both P and Q and $P \times Q$ is rhs



Cross Product Details

- How do we make sure \mathbf{N} is an *outward* normal?
 - Order vertices of the polygon to make it so
 - As viewed from outside the solid object, order vertices in counter-clockwise order
- $\|\mathbf{N}\| = (\|P\| \|Q\| \sin \beta)$ is the area of the parallelogram formed by P and Q
- β is the angle between P and Q
- Collinear vertices test?
 - Could just test $\|\mathbf{N}\|$
 - if 0, move to next vertex
 - stop when loop around



Depth Buffer Visible Surface Algorithm

- Brute force image precision algorithm
 - Polygon *scan conversion* determines all pixels that are “inside” the bounds of a polygon and assigns the polygon color to them
 - Uses a 2D *color* array that is *xResolution* by *yResolution*; *color[y,x]* stores the color to be assigned to the pixel at *(x,y)*.
 - To add visibility to this algorithm
 - Define a 2D *depth* buffer that is the same size as the *color* buffer; *depth[y,x]* stores the z-value of the last polygon whose color is in *pixel[y,x]*.
- This is a standard feature on modern graphics cards
 - requires 3D coordinates and perspective transformation

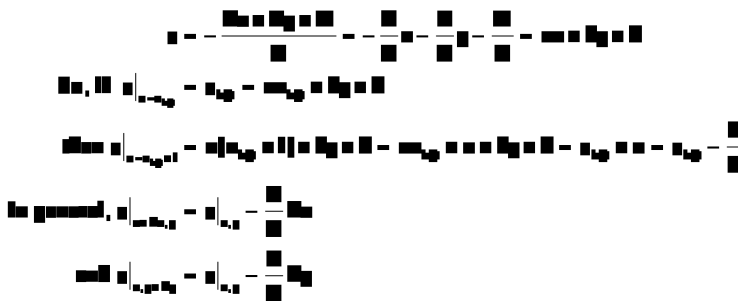
```
writePixel(x,y,z,color)
{
  if ( z < depth[y,x] )
    depth[y,x] = z
    color[y,x] = color
}
```

Depth Buffer Issues

- What if z-values are the same?
 - Doesn't matter which you choose -- it'll be wrong at least half the time!
 - If there were truly 2 different objects at the same place, the scene would be invalid
 - Round-off error, however, is a real issue
 - Consider polygons that share an edge: one is a back face and backface culling has not been applied!
 - It will seem like a random decision as to which z value along the edge is determined to be "in front", so you'll get scattered pixels along the edge of the front face with the color of the otherwise invisible back face.

Depth Coherence

- Faces are planar; z-values change linearly across a face.
 - Once we compute a z-value at the left end of a scan line, subsequent z-values for neighboring pixels are cheaper



Review