Overview

- Rendering topics
  - Z-buffering
  - Back-Face Culling
  - Ray Tracing (Ray Casting)

Mesh/Faceted Model

Back-Face Culling

- Assumptions:
  - Object approximated as closed polyhedron
  - Polyhedron interior is not exposed by the front cutting plane
  - Eye-point not inside object
  - Right-hand vertex ordering defines outward normal
  - Polygons not facing the viewer called Back-Facing
- Back-Face Culling is a technique for eliminating polygons for these kinds of models
- On average eliminates half of the polygons
  - Could be done for performance reasons

Back-Face Culling

- After canonical transformation, examine normal \( \mathbf{N}_k \) \((x_k, y_k, z_k)\) to the face.
- If \( z_k < 0 \), face is a Back-Face - don’t draw it
- More general test looks at \( \mathbf{N}_k \cdot \mathbf{V} \)
  - \( \mathbf{V} \) - View vector
- The only test necessary for a single convex polyhedron

Normal for Triangle

plane \( \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \)

\[ \mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \]

normalize \( \mathbf{n} \rightarrow \mathbf{n} / ||\mathbf{n}|| \)

Note that right-hand rule determines outward face
Back-Face Culled Wire-Frame

Z-buffering

- Z-buffering (depth-buffering) is a visible surface detection algorithm
- Implementable in hardware and software
- Requires data structure (z-buffer) in addition to frame buffer.
- Z-buffer stores values [0 .. ZMAX] corresponding to depth of each point.
- If the point is closer than one in the buffers, it will replace the buffered values

Z-buffering

for (y = 0; y < YMAX; y++)
for (x = 0; x < XMAX; x++) {
  ... for each polygon...
  Z[x][y] = ZMIN;
  F[x][y] = BACKGROUND_COLOR;
  for (each pixel in polygon’s projection) {
    pz = polygon’s z-value at pixel coordinates (x,y)
    if (pz > Z[x][y]) { /* New point is closer */
      Z[x][y] = pz;
      F[x][y] = polygon’s color at pixel coordinates (x,y)
    }
  }
}

Z-buffering w/ front/back clipping

for (y = 0; y < YMAX; y++)
for (x = 0; x < XMAX; x++) {
  ... for each polygon...
  for (each pixel in polygon’s projection) {
    pz = polygon’s z-value at pixel coordinates (x,y)
    if (pz < FRONT && pz > Z[x][y]) { /* New point is behind front plane & closer than previous point */
      Z[x][y] = pz;
      F[x][y] = polygon’s color at pixel coordinates (x,y)
    }
  }
}

Z Interpolation

- We can simplify the calculation of z by exploiting the fact that triangle is planar.
  - Interpolate z values along the edges
  - Interpolate z values along scan line
  - Special cases: horizontal edge, degenerate triangle & single vertex
\[ z_a = z_1 + \frac{|P_a - P_1|}{|P_2 - P_1|}(z_2 - z_1) \]
\[ z_b = z_1 + \frac{|P_b - P_1|}{|P_3 - P_1|}(z_3 - z_1) \]
\[ z_p = z_1 + \frac{|P_p - P_a|}{|P_b - P_a|}(z_2 - z_a) \]

- \( P_1 = (x_1, y_1) \)
- \( P_2 = (x_2, y_2) \)
- \( P_3 = (x_3, y_3) \)

**Depth Cueing**
- Objects that are closer are brighter
- Objects farther away are darker
- Color = BaseColor \( \times (z - \text{far})/(\text{near} - \text{far}) \)

See the Difference

Back-Face Culled & Z-Buffered Wire-Frame

Depth Cueing

www.dma.unibo.it/~casciola

www.siggraph.org
Ray Casting (Ray Tracing)

- Determines visible surfaces by tracing rays of light from the viewer's eye to the objects in the world.

Ray Casting

- Determines visible surfaces by tracing rays of light from the viewer's eye to the objects.
- View plane is divided on the pixel grid.
- The eye ray is fired from the center of projection through each pixel.

Computing Intersections with Sphere

- Sphere with radius $r$ and center $(a,b,c)$ is:
  \[ (x-a)^2 + (y-b)^2 + (z-c)^2 = r^2 \]
- Intersection is found by substituting values for $(x,y,z)$:
  \[ (x + t\Delta x - a)^2 + (y + t\Delta y - b)^2 + (z + t\Delta z - c)^2 = r^2 \]
- With some straightforward algebraic transformations:
  \[ \Delta x^2 + \Delta y^2 + \Delta z^2 \cdot t^2 + 2(\Delta x(x-a) + \Delta y(y-b) + \Delta z(z-c))t + (a^2 + b^2 + c^2 - r^2) = 0 \]
- Equation is quadratic in terms of $t$.
- Solve with quadratic formula:
  \[ t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \]
- No real roots (square root is negative). No intersection.
- One real root. Ray grazes the sphere.
- Two real roots. There are two points of intersection.

Relation of $t$ to Intersection

We want the smallest positive $t$ - call it $t_i$.
Ray-triangle Intersection

• Insert ray equation into barycentric expression of triangle
• \( P(t) = a + \beta (b-a) + \gamma (c-a) \)
• Intersection if \( \beta + \gamma < 1, \ 0 < \beta \) and \( 0 < \gamma \)

Computing Intersections with Polygon

• First intersect ray with plane \( Ax + By + Cz + D = 0 \)
• The substitution results in \( t = \frac{Ax_0 + By_0 + Cz_0 + D}{A\beta + B\gamma + C\delta} \)
• If denominator is 0, ray is parallel to the plane
• Project polygon and point orthographically on the coordinate plane
• Polygon containment test can be performed in 2D

Polygon Containment Test

• Jordan Curve Theorem:
  – Point is inside if, for any ray, there is an odd number of crossings
  – Otherwise it is outside
• Be careful with all the special cases
• Wide variety of other techniques exist

Why Trace Rays?

• More elegant
• Testbed for techniques:
  – modeling (reflectance, transport)
  – rendering (e.g. Monte Carlo)
  – texturing (e.g. hypertexture)
• Easiest photorealistic renderer to implement

Ray Tracing

• Extension of ray casting
• Idea: Continue to bounce the ray in the scene
• Shoot rays to light sources
• Simple and powerful
• Reflections, shadows, transparency and multiple light sources
• Can be used to produce highly realistic images

Ray Traced Image
Snell's Law: \( \eta_i \sin \theta_i = \eta_t \sin \theta_t \)

Let \( \eta = \eta_t/\eta_i = \sin \theta_t/\sin \theta_i \)

Let \( m = (\cos \theta \mathbf{n} - \mathbf{i}) / \sin \theta \)

Then...

\[
t = \sin \theta m - \cos \theta \mathbf{n} = (\sin \theta / \sin \theta_i) (\cos \theta \mathbf{n} - \mathbf{i}) - \cos \theta \mathbf{n} = (\eta \cos \theta - \cos \theta_i) \mathbf{n} - \eta \mathbf{i}
\]

\[
t = \left( \eta (\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - \eta^2 (1 - (\mathbf{n} \cdot \mathbf{i})^2)} \right) \mathbf{n} - \eta \mathbf{i}
\]

\[
\cos \theta_i = \sqrt{1 - \sin^2 \theta_i}
\]

Can be negative for grazing angles where \( \eta_i < 1 \), ray which going from glass to air, resulting in total internal reflection (no refraction).

Efficiency Considerations

- Partition the bounding box on a regular grid with equal sized extents
- Associate each partition with the list of objects contained in it
- Ray only intersected with the objects contained in the partitions it passes through
- If partitions are examined in the order ray travels, we can stop after first intersection is found
- Need to check if intersection itself is in the partition