Hidden Surface Removal

Visibility

- Assumption: All polygons are opaque
- What polygons are visible with respect to your view frustum?
  - Outside: View Frustum Clipping
    - Remove polygons outside of the view volume
    - For example, Liang-Barsky 3D Clipping
  - Inside: Hidden Surface Removal
    - Backface culling
    - Polygons facing away from the viewer
    - Occlusion
    - Polygons farther away are obscured by closer polygons
    - Full or partially occluded portions
- Why should we remove these polygons?
  - Avoid unnecessary expensive operations on these polygons later

No Lines Removed

Hidden Lines Removed
Hidden Surfaces Removed

Hidden Surface Removal

Occlusion: Full, Partial, None

- The rectangle is closer than the triangle
- Should appear in front of the triangle

Backface Culling

- Avoid drawing polygons facing away from the viewer
  - Front-facing polygons occlude these polygons in a closed polyhedron
- Test if a polygon is front- or back-facing?

Detecting Back-face Polygons

- The polygon normal of a ...
  - front-facing polygon points towards the viewer
  - back-facing polygon points away from the viewer

If \((n \cdot v) > 0\) ⇒ “back-face”
If \((n \cdot v) \leq 0\) ⇒ “front-face”
\(v = \text{view vector}\)

- Eye-space test … EASY!
  - “back-face” if \(n_z < 0\)

- \text{glCullFace}(GL\_BACK)
**Polygon Normals**

- Let polygon vertices $v_0, v_1, v_2, \ldots, v_{n-1}$ be in counterclockwise order and co-planar.
- Calculate normal with cross product:
  $$\mathbf{n} = (v_1 - v_0) \times (v_{n-1} - v_0)$$
- Normalize to unit vector with $\mathbf{n}/\|\mathbf{n}\|$.

**Normal Direction**

- Vertices counterclockwise $\Rightarrow$ Front-facing
- Vertices clockwise $\Rightarrow$ Back-facing

**Painter's Algorithm (1)**

- Assumption: Later projected polygons overwrite earlier projected polygons.

**Painter's Algorithm (2)**

- **Main Idea**
  - A painter creates a picture by drawing background scene elements before foreground ones.
- **Requirements**
  - Draw polygons in back-to-front order.
  - Need to **sort** the polygons by depth order to get a correct image.
Painter’s Algorithm (3)
- Sort by the depth of each polygon

Painter’s Algorithm (4)
- Compute $z_{\min}$ ranges for each polygon
- Project polygons with furthest $z_{\min}$ first

Painter’s Algorithm (5)
- Problem: Can you get a total sorting?

Painter’s Algorithm (6)
- Cyclic Overlap
  - How do we sort these three polygons?
  - Sorting is nontrivial
    - Split polygons in order to get a total ordering
    - Not easy to do in general
Visibility

- How do we ensure that closer polygons overwrite further ones in general?

Z-Buffer

- Depth buffer (Z-Buffer)
  - A secondary image buffer that holds depth values
  - Same pixel resolution as the color buffer
  - Why is it called a Z-Buffer?
    - After eye space, depth is simply the z-coordinate

- Sorting is done at the pixel level
  - Rule: Only draw a polygon at a pixel if it is closer than a polygon that has already been drawn to this pixel

Z-Buffer Algorithm

- Visibility testing is done during rasterization

```c
for (each face F)
    for (each pixel (x,y) covering the face)
    {
        depth = depth of F at (x,y);
        if (depth < d[x][y])       //F is closest so far
            c = color of F at (x, y);
             set the pixel color at (x, y) to c;
        d[x][y] = depth; // update the depth buffer
    }
```

Z-buffer: A Secondary Buffer

- Color buffer
- Depth buffer
Z-Buffer

- How do we calculate the depth values on the polygon interior?

\[
\begin{align*}
Z_a &= Z_i + (Z_4 - Z_i) \frac{(y_i - y_3)}{(y_4 - y_3)} \\
Z_b &= Z_i + (Z_2 - Z_i) \frac{(y_i - y_4)}{(y_2 - y_4)} \\
Z_p &= Z_i + (Z_b - Z_a) \frac{(x_i - x_p)}{(x_b - x_p)} \\
\end{align*}
\]

Bilinear Interpolation
Z-Buffer Algorithm

- Algorithm easily handles this case

---

Z-buffering in OpenGL

- Create depth buffer by setting GLUT_DEPTH flag in glutInitDisplayMode() or the appropriate flag in the PIXELFORMATDESCRIPTOR.

- Enable per-pixel depth testing with glEnable(GL_DEPTH_TEST)

- Clear depth buffer by setting GL_DEPTH_BUFFER_BIT in glClear()