Lecture 13: Visibility

Rui Wang
Visible Surfaces

- Problem: Given a set of 3D objects and a viewing condition, determine which lines or surfaces are visible
- Called *hidden surface/line removal*
Painter’s Algorithm

- Draw objects in order (far to near)
Painter’s Algorithm

- The order changes view-dependently
Painter’s Algorithm

☐ Problem
BSP Tree

- Binary Space Partitioning (BSP)
  - A single data structure that is useful for any viewpoint
  - Requires some preprocessing, assumes the scene is static
BSP: The Basic Idea

1. Find a plane that separates the scene into two disjoint sets of polygons

\[ f(x, y, z) = ax + by + cz + d = 0 \]
BSP: The Basic Idea

2. Check which side the eye point falls in, draw triangles in depth order (from far to near)

if \( f(e) < 0 \) then
draw \( T_1 \)
draw \( T_2 \)
else
draw \( T_2 \)
draw \( T_1 \)

\[ f(x, y, z) = ax + by + cz + d = 0 \]
BSP: The Basic Idea

1. Select any triangle -> current splitting plane
2. Partition the remaining triangles into two groups, based on their position relative to the “splitting” plane (use plane normal)
3. If a triangle is in both groups (intersects the plane), split it along the plane
4. Repeat on the “front” group
5. Repeat on the “back” group
BSP Tree Example

One constructed BSP tree

A different construction
function draw(bsptree node, point e)
if (node.is_empty()) then
    return
if (node.plane_equation(e) < 0) then
    draw(node.front_child, e)
rasterize node.triangle
draw(node.back_child, e)
else
    draw(node.back_child, e)
rasterize node.triangle
draw(node.front_child, e)
BSP Tree Facts

- Starting at the root, make sure we draw everything in the right order.
  - Draw everything further from the eye first
  - Draw the splitting polygon
  - Draw everything closer to the eye

- Optimizing the Tree
  - Balancing is not a concern. Why?
  - What about splitting?
The Z-Buffer Algorithm

- Universally implemented in hardware
- Resolve visibility in screen space, after fragments are generated
- Very simple to implement
The Z-Buffer Algorithm

- Use an additional buffer to hold depth values:
  - Render primitives in arbitrary order
  - Record their depths in the depth buffer
  - If the depth of a pixel about to be drawn is greater than what’s already there, throw it away

```c
function drawpixel(int x, int y, color c, float z)
if (z < z_buffer[x, y]) then
    z_buffer[x,y]=z;
    frame_buffer[x,y]=c;
```

Z Buffer Example
Z Buffer Example
Why Is Z-Buffer So Popular?

- Advantages
  - Simple to implement in hardware
  - One more chunk of memory
  - One more interpolator
  - One more comparison
  - Image precision
  - Supports non-polygonal primitives
  - Unlimited scene complexity
  - No preprocessing, dynamic Scenes
  - Depth values can be saved for later use
Why Is Z-Buffer So Popular?

- Disadvantages
  - Extra memory and bandwidth
  - Waste time drawing hidden objects
  - Z precision errors lead to depth aliasing
    - Integer Z values
Why is Z-Buffer So Popular?

- **Disadvantages**
  - Extra memory and bandwidth
  - Waste time drawing hidden objects
  - Z precision errors lead to depth aliasing
  - Integer Z values

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & f+n & -f \cdot n \\
0 & 0 & 1 & 0 \\
\end{pmatrix}
\begin{bmatrix}
x_w \\
y_w \\
z_w \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
\begin{bmatrix}
\frac{x}{z} \\
\frac{y}{z} \\
(f+n)\frac{-f \cdot n}{z} \\
1 \\
\end{bmatrix}
\]
Why is Z-Buffer So Popular?

- **Disadvantages**
  - Extra memory and bandwidth
  - Waste time drawing hidden objects
  - Z precision errors lead to depth aliasing

- Integer Z values

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & f + n & -f \cdot n \\
0 & 0 & 1 & 0
\end{pmatrix}
\begin{bmatrix}
x_w \\
y_w \\
z_w \\
1
\end{bmatrix}
= \begin{bmatrix}
x/z \\
y/z \\
(f+n) \cdot f \cdot n/z \\
1
\end{bmatrix}
\]

\[d = (f + n) - \frac{f \cdot n}{z_w}\]

Maps \(n\)-\(n\), \(f\)-\(f\)
Precision of Z-Buffer

- Depth value $d$ stored as N-bit integer

1-bit variation

$$\Delta z_w = \frac{f \cdot n \cdot \Delta d}{((f + n) - d)^2} = \frac{z_w^2 \cdot \Delta d}{f \cdot n}$$

$$\Delta d = \frac{(f - n)}{2^N - 1}$$

$$\Delta z_{w_{\text{min}}} = \frac{n \cdot \Delta d}{f}$$

$$\Delta z_{w_{\text{max}}} = \frac{f \cdot \Delta d}{n}$$

Rule of Thumb: To reduce depth buffer precision problem, choose $f$ as close as possible, and $n$ as far as possible.
Basics about OpenGL Z-Buffer

☐ Turning on Z-Buffer / Depth Buffer
   
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH);

☐ Reset Z-Buffer
   
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

☐ Enable/Disable Z-test
   
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LESS);
Back Face Culling

Red edges are not drawn
Green edges are drawn

Viewing direction
Back Face Culling

Red edges are not drawn
Green edges are drawn

Silhouettes Edge?

Viewing direction
Hidden Line Removal

How to achieve hidden line removal?
Hidden Line Removal

How to achieve hidden line removal?

Draw the scene twice, once in solid polygon mode, once in wireframe mode; reuse the z-buffer.