CS4533 Lecture 8 Slides/Notes

Hidden Surface Removal & BSP Trees;
Shadow Projection & Making Decal in HW3
(Notes, Ch 11, Notes)

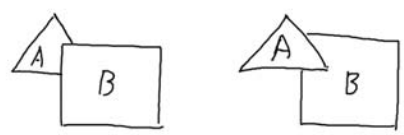
By Prof. Yi-Jen Chiang
CSE Dept., Tandon School of Engineering
New York University

Hidden Surface Removal:

When drawing opaque objs.
the portions that are occluded
by closer portions of other objs
should be hidden / removed.

* object-space vs. image-space methods

object-space Method:



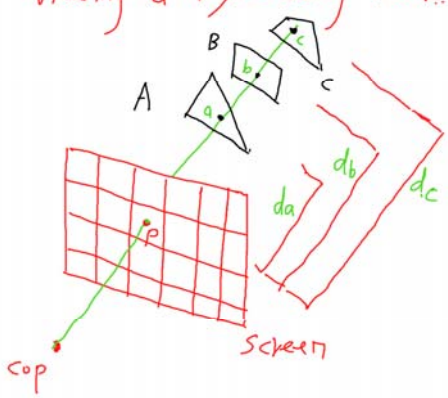
Consider polygon pairs.

K polygons $\Rightarrow O(K^2)$ pairs
$O(K^2)$ computation.

Good for small K only

Image-Space Method:

Viewing & ray-casting ideas.



The color of pixel p should be the color of pt a.

* Fragments a, b, c are all rasterized to pixel p.
compare their distances d_a, d_b, d_c.
d_a is the smallest. i.e. a is closest to the eye.
So finally a is drawn to pixel p.

Display: $m \times n$. K polygons.

$\Rightarrow O(mn \cdot K)$ m, n are fixed constants.

$= O(K)$ computation. Good for large K.

Method of choice: <u>Z-buffer algorithm.</u>

(Hardware support.
Image-Based Method.)

The Z-buffer Algorithm: the method of choice for hidden surface removal.

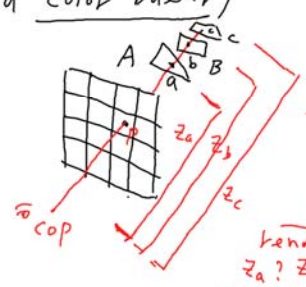
For pixel p:
z_p: value in z-buffer
c_p: color in frame buffer

It works in the image space. yet loops over polygons, combined with rasterization.

There is a <u>z-buffer</u> (also called the <u>depth buffer</u>) with the same resolution as the <u>frame buffer</u> (also called <u>color buffer</u>)

e.g. 1248 × 1024 pixels in frame buffer
⇓
1248 × 1024 elements in z-buffer.

Each element stores the distance between the cop and the closest fragment so far of the corresponding pixel.



render B.
$z_p \leftarrow z_b$
$c_p \leftarrow c_b$.

render C
z_c ? z_p. ($z_c > z_p$)
Ignore fragment c.

render A.
z_a ? z_p ($z_a < z_p$)
$z_p \leftarrow z_a$, $c_p \leftarrow c_a$.

Combined with rasterization : Rasterize scan-line by scan line.

Suppose polygon is on the plane : $ax + by + cz + d = 0$.

$(x_1, y_1, z_1) \longrightarrow (x_2, y_2, z_2)$ $\quad \Delta x = x_2 - x_1$, $\quad a\Delta x + b\Delta y + c\Delta z = 0$.

current pixel $\quad$ next pixel. $\quad \Delta y = y_2 - y_1$ $\quad$ scan line is horizontal $\Rightarrow \Delta y = 0$

$\Delta z = z_2 - z_1$, $\quad \Delta z = \dfrac{-a\Delta x}{c}$

But we move one pixel to the right.

$\Rightarrow \Delta x$ is a constant.

$\Rightarrow \Delta z$ is a constant. computed once for each polygon.

As we move left to right along a scan line.

we can incrementally update the z-value by the constant Δz.

OpenGL commands for z-buffer alg. :

in init()

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
```

double buffering $\quad$ RGB color mode $\quad$ z buffer

```
glEnable(GL_DEPTH_TEST);
```

enable z-buffer testing

in display(). at the beginning of each frame.

```
glClear(GL_DEPTH_BUFFER_BIT);
glClear(GL_COLOR_BUFFER_BIT);
```

clear the z-buffer.

" " frame buffer

Occlusion culling

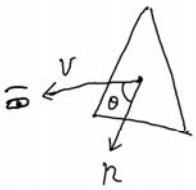
* z-buffer alg. is the base-line approach.

On top of that, we want to have an algorithmic method to detect occlusion early to avoid sending hidden/occluded polygons to the pipeline for rendering

[Below we look at occlusion culling Algorithms.]

1. Back-Face Removal:

Suppose we have closed surface of an obj (e.g. sphere, cube, ...), where each polygonal face has a normal vector going outward. e.g.



polygon is facing forward iff $\theta \in [-90^\circ, 90^\circ]$

i.e. $\cos\theta \geq 0$.

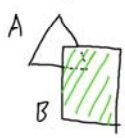
i.e. $n \cdot v \geq 0$. $(n \cdot v = |n||v| \cos\theta)$

// iff $n \cdot v \geq 0$.

If $(n \cdot v \geq 0)$ render the polygon; otherwise, don't.

2. Depth Sort and Painter's Algorithm.

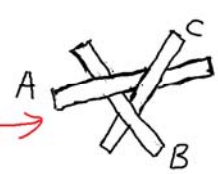
Draw polygons from the farthest to the closest (back to front)



The overlapped portions are over-written.
Only the closest portions are retained.

⇒ We need Depth Sort (sort polygons from the farthest to the closest)

Problem: We may have cyclic ordering. e.g.



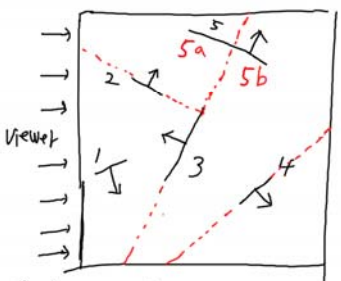
i.e. Depth order may NOT exist

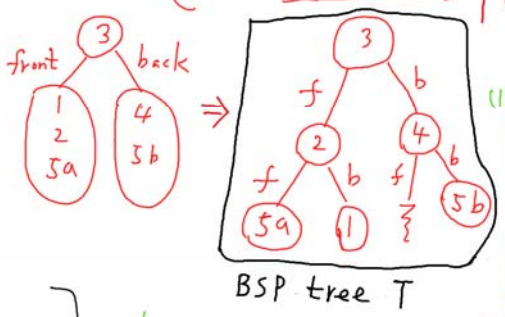
Partial Solution: break into pieces.

Q: How do we break?
A: space-partitioning tree such as BSP tree.

BSP Tree (Binary-Space Partition tree)

(Render back-to-front)

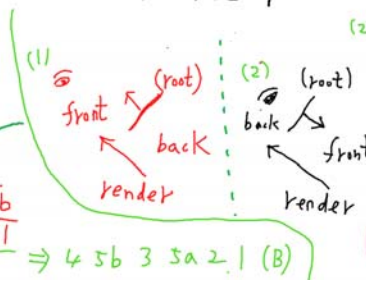




BSP tree T

```
BSP_Render (root)
{
  // Base case for recursion
  If (root == NULL) return;
  (1) If (viewer is in front of root)
  {
    BSP_Render(back_child(root));
    display (root);
    BSP_Render(front_child(root));
  }
  (2) else // viewer is behind root
  {
    BSP_Render (front_child (root));
    display (root);
    BSP_Render (back_child (root));
  }
}
```

(polygons 1 ~ 5 are projected onto the 2D plane as line segments. ↑ indicates the front side.)



Using BSP_Render( ) on T:

(A) 4 5b 3 1 2 5a

(B) 4 5b 3 5a 2 1

(C) 5a 2 1 3 4 5b.

⇒ 4 5b 3 5a 2 1 (B)

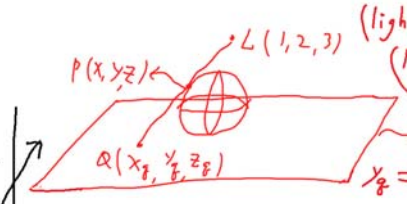
HW3 Part (a): Shadow Projection

LookAt( ) · (rollinp_transf) · Sphere( )

⇒ shadow:

LookAt( ) · (shadow_proj) · (rollinp_transf) · Sphere( )

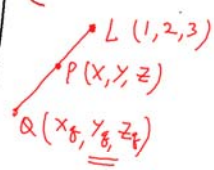
(3) (2) (1)



(light at L. eg. L = (1,2,3))

(P(x,y,z): any generic point on the sphere.)

* Express (x_q, y_q, z_q) in terms of x, y, z.
Then derive matrix M. st. $(M \cdot P = Q)$



$$\frac{x_q - 1}{x - 1} = \frac{y_q - 2}{y - 2} = \frac{z_q - 3}{z - 3} \quad (y_q = 0)$$

(1) (2)

From (1). $x_q = \left(\frac{y_q - 2}{y - 2}\right)(x - 1) + 1$

From (2) $z_q = \left(\frac{y_q - 2}{y - 2}\right)(z - 3) + 3$.

$y_q = 0$.

$$\begin{cases} x_q = \dfrac{f(x,y,z)}{\boxed{y-2}} \\ z_q = \dfrac{h(x,y,z)}{\boxed{y-2}} \\ y_q = 0. \end{cases}$$

$\Rightarrow (x, y, z) \xrightarrow{M} (x_q, y_q, z_q)$, $M = ?$

* Key: Use Homogeneous Coordinate System!

$$\overset{Q}{\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix}} = \begin{bmatrix} \frac{f(x,y,z)}{y-2} \\ 0 \\ \frac{h(x,y,z)}{y-2} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f(x,y,z) \\ 0 \\ h(x,y,z) \\ y-2 \end{bmatrix} = \underbrace{\begin{bmatrix} \text{constant entries} \\ \text{independent} \\ \text{of } x, y, z \\ 4\times 4 \end{bmatrix}}_{M} \overset{P}{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}$$

(Perspective Division (P.D.))

Recall:

$w \neq 0, w \neq 1$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \underset{(P.D.)}{\equiv} \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

* Note: The 16 entries in the 4×4 matrix M must be constants, independent of x, y, z, so that M can be applied to all different points (with different x, y, z) on the sphere.

* HW3 Part (b): Making Decal

shadow is the decal on top of the ground

There are 2 buffers: frame buffer & z-buffer.

we can enable/disable writing to these buffers.

0. Always enable z-buffer testing
1. Disable writing to z-buffer.
   Draw ground (only to frame buffer). NOT to z-buffer.
2. Enable writing to z-buffer.
   Draw shadow (to both buffers)

* (ground is NOT in z-buffer to block shadow. so shadow is always drawn on top of ground.)

(Now, We want to put back ground to z-buffer while preserving the current content of frame buffer i.e. draw ground only to z-buffer.)

3. Disable writing to frame buffer.
   Draw ground. (only to z-buffer)
4. Enable writing to frame buffer.
   Resume normal ops.

⌈ ⌋ : critical section.

Any other objs should be drawn outside ⌈ ⌋