CS4533 Lecture 7 Slides/Notes

Hidden Surface Removal (Notes, Ch 11)

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

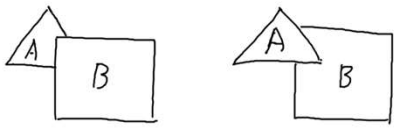
1

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

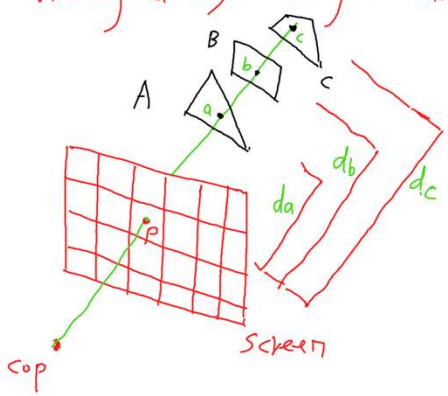
$O(K^2)$ computation.

Good for small K only

2

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: Z-buffer algorithm.

(Hardware support.
Image-Based Method.)

3

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.

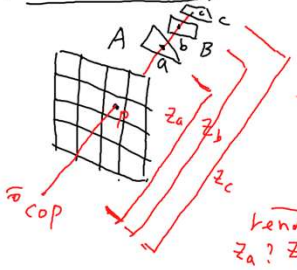
There is a z-buffer (also called the depth buffer) with the same resolution as the frame buffer (also called color buffer)

eg. 1248 × 1024 pixels in frame buffer

⇓

1248 × 1024 elements in z-buffer.

Each element stores the distance between the eye 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 ($z_a < z_p$)
$z_p \leftarrow z_a$, $c_p \leftarrow c_a$.

4

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

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

$(x_1, y_1, z_1) \longrightarrow (x_2, y_2, z_2)$

current pixel next pixel.

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

$\Delta z = z_2 - z_1$

$a\Delta x + b\Delta y + c\Delta z = 0$.

scan line is horizontal $\Rightarrow \Delta y = 0$

$\Delta z = -\frac{a}{c}\Delta x$

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.

5

OpenGL commands for z-buffer alg.:

in init():

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

double buffering RGB color mode z buffer

```
glEnable(GL_DEPTH_TEST);
```

enable z-buffer testing

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

```
glClear(GL_DEPTH_BUFFER_BIT);   clear the z-buffer.
glClear(GL_COLOR_BUFFER_BIT);   //  //  frame buffer
```

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

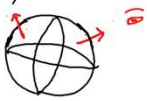
On top of that, we want to have an algorithic method to detect occlusion early to avoid sending hidden/occluded polygons to the pipeline for rendering

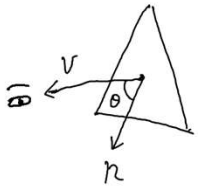
Occlusion culling

6

[Below we look at occlusion culling Algorithms.]

1. Back-Face Removal:

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



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.