CS6533/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.

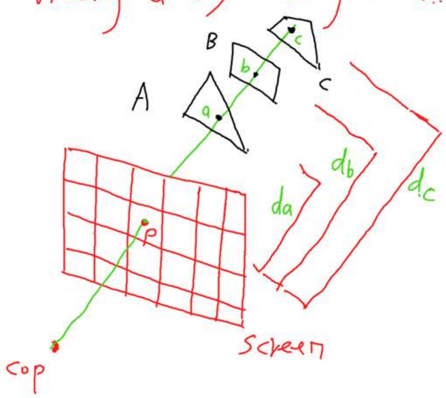
K polygons $\Rightarrow O(K^2)$ pairs
$O(K^2)$ computation.

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.

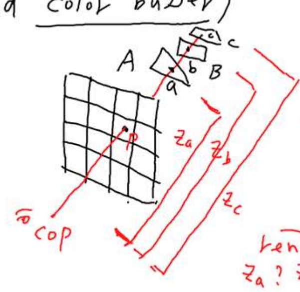
There is a z-buffer (also called the depth buffer) with the same resolution as the frame buffer (also called color buffer)

e.g. 1248×1024 pixels in frame buffer

$\Downarrow$

1248×1024 elements in z-buffer.

Each element stores the distance between the cop and the closest frament 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 : $aX + bY + cZ + d = 0$.

$(X_1, Y_1, Z_1) \longrightarrow (X_2, Y_2, Z_2)$ $\Delta X = X_2 - X_1$, $a\Delta X + b\Delta Y + c\Delta Z = 0$.

current pixel next pixel. $\Delta Y = Y_2 - Y_1$ Scan line is horizontal $\Rightarrow \Delta Y = 0$

$\Delta Z = Z_2 - Z_1$ $\Delta Z = \frac{-a\Delta X}{c}$

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.

5

OpenGL commands for z-buffer alg. :

in init()

glutInitDisplayMode ($\underbrace{\text{GLUT\_DOUBLE}}_{\text{double buffering}}$ | $\underbrace{\text{GLUT\_RGB}}_{\text{RGB color mode}}$ | $\underbrace{\text{GLUT\_DEPTH}}_{\text{z buffer}}$);

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

Occlusion culling

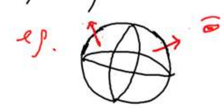
* z-buffer alg. is the base-line approach.
On top of that, we want to have an algorithmic method to detect occlusion early to avoid sending hidden/occluded polygons to the pipeline for rendering

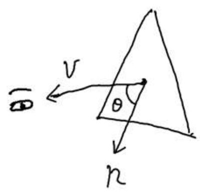
6

[Below we look at occlusion culling Algorithms.]

1. <u>Back-Face Removal</u>:

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





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

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

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

iff $n \cdot v \geq 0$.

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

7