

# Computer Graphics (CS 543)

## Lecture 10 (Part 2): Viewport Transformation & Hidden Surface Removal

Prof Emmanuel Agu

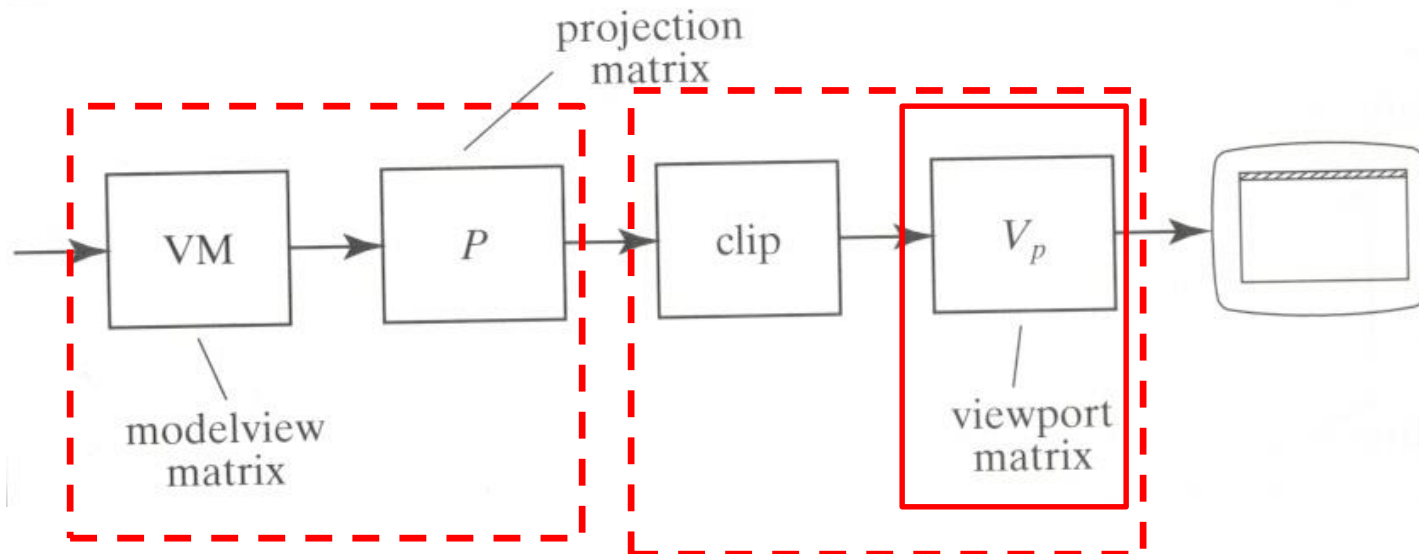
*Computer Science Dept.  
Worcester Polytechnic Institute (WPI)*





# Viewport Transformation

- After clipping, do viewport transformation



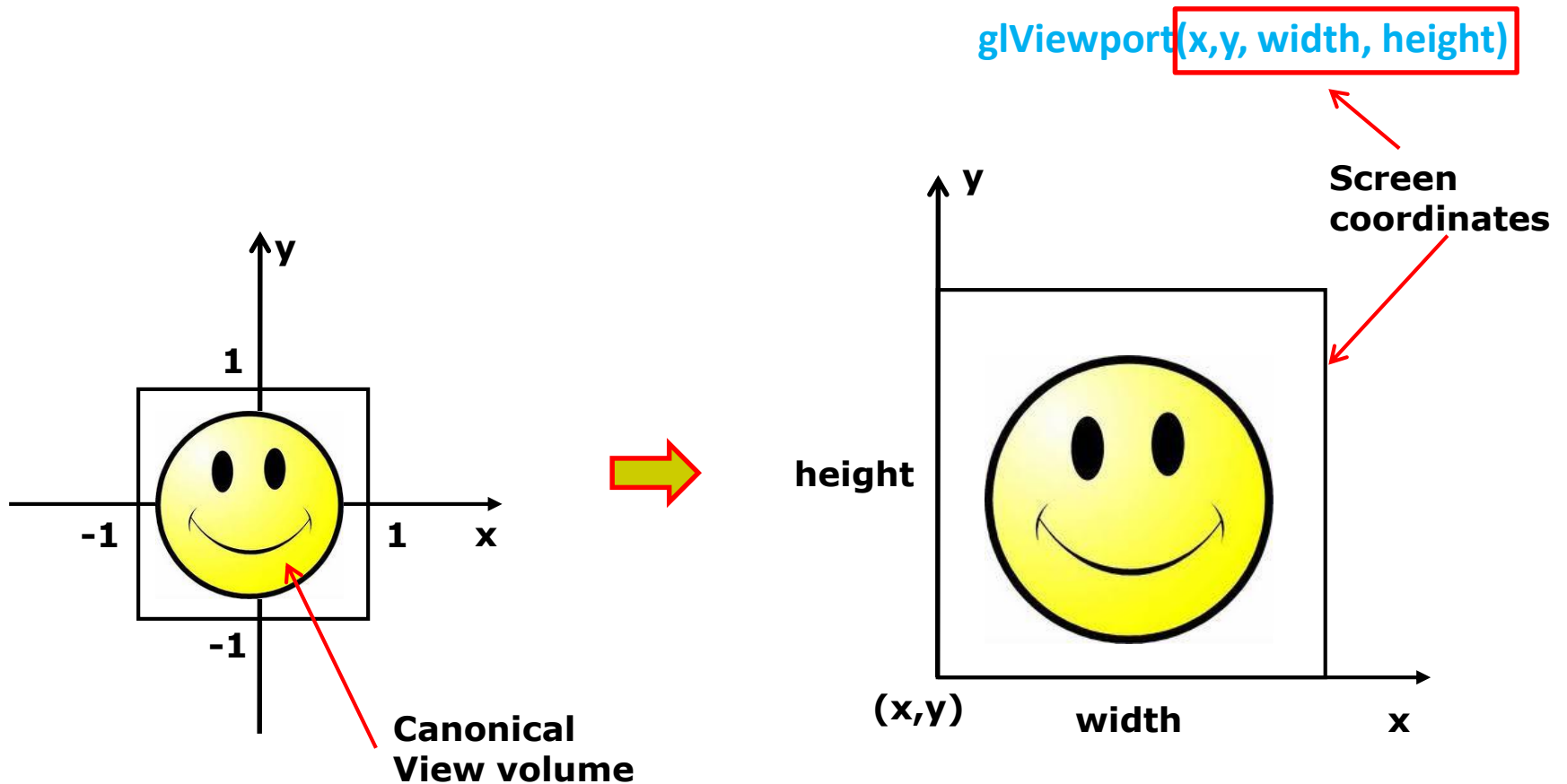
**User implements in  
Vertex shader**

**Manufacturer  
implements  
In hardware**



# Viewport Transformation

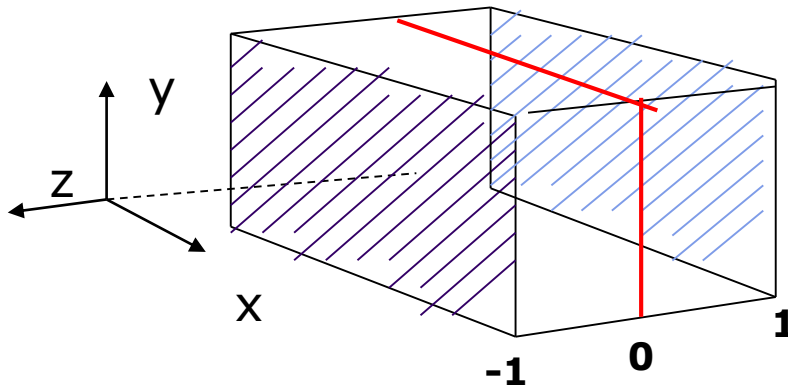
- Maps **CVV (x, y)** -> **screen (x, y)** coordinates





# Viewport Transformation: What of z?

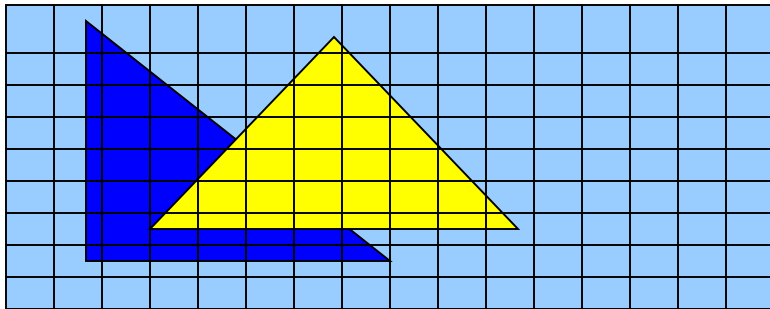
- Also maps  $z$  (pseudo-depth) from  $[-1,1]$  to  $[0,1]$
- $[0,1]$  pseudo-depth stored in depth buffer,
  - Used for Depth testing (Hidden Surface Removal)





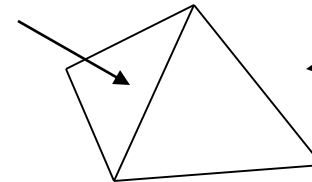
# Hidden surface Removal

- Drawing polygonal faces on screen consumes CPU cycles
- User cannot see every surface in scene
- To save time, draw only surfaces we see
- Surfaces we cannot see and elimination methods?



**1. Occluded surfaces:** hidden surface removal (visibility)

Back face

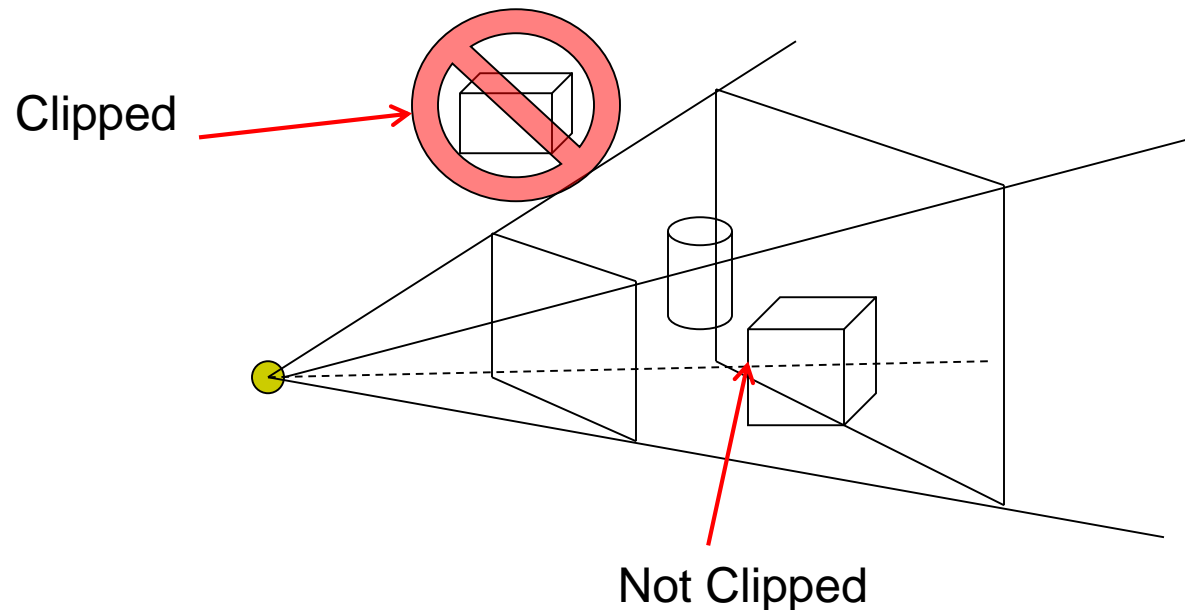


**2. Back faces:** back face culling

# Hidden surface Removal



- Surfaces we cannot see and elimination methods:
  - **3. Faces outside view volume:** viewing frustum culling



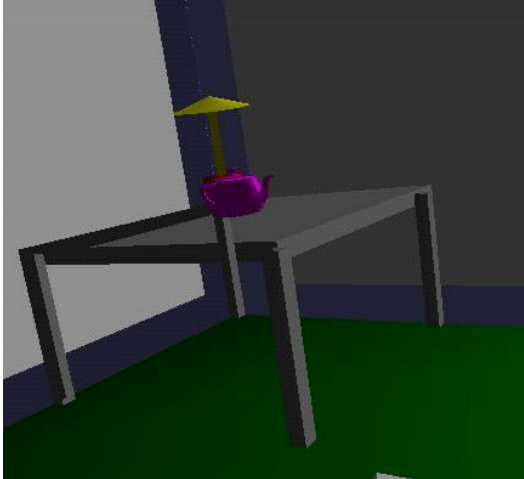
Classes of HSR techniques:

- **Object space techniques:** applied before rasterization
- **Image space techniques:** applied after vertices have been rasterized

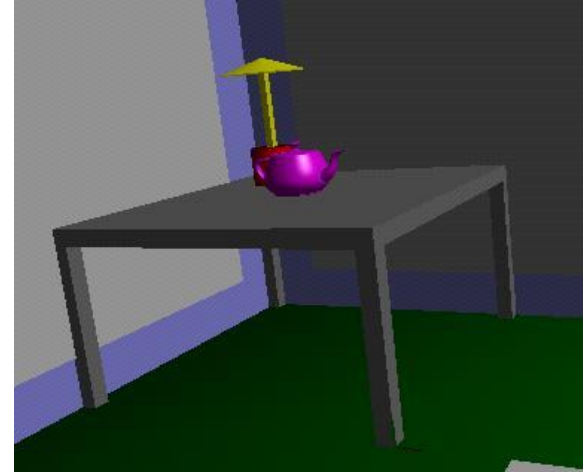


# Visibility (hidden surface removal)

- Overlapping opaque polygons
- **Correct visibility?** Draw only the closest polygon
  - (remove the other hidden surfaces)



**wrong visibility**

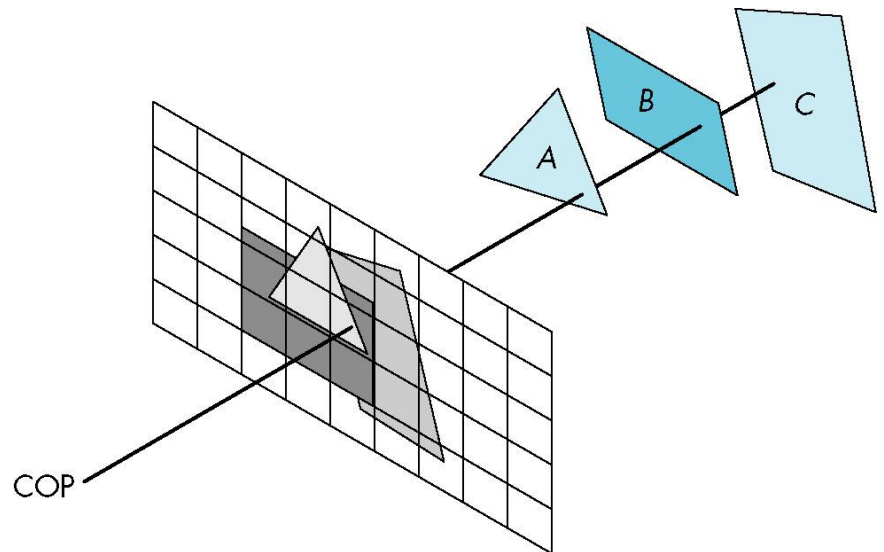


**Correct visibility**



# Image Space Approach

- Start from pixel, work backwards into the scene
- Through each pixel, ( $nm$  for an  $n \times m$  frame buffer) find closest of  $k$  polygons
- Complexity  $O(nmk)$
- Examples:
  - Ray tracing
  - z-buffer : OpenGL



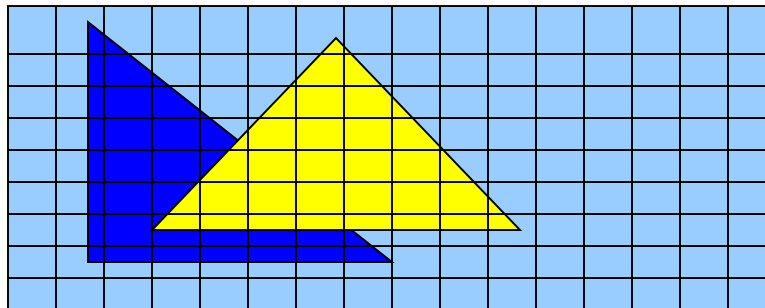




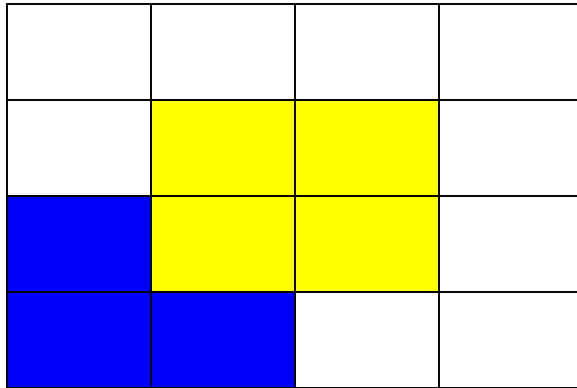
# OpenGL - Image Space Approach

- Paint pixel with color of **closest** object

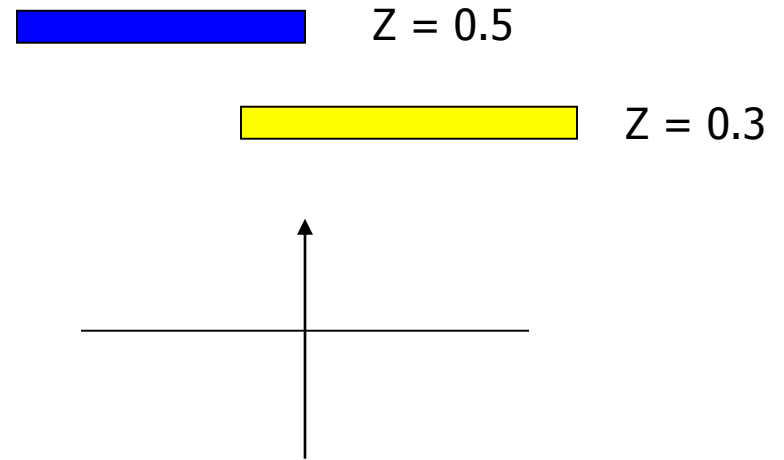
```
for (each pixel in image) {  
    determine the object closest to the pixel  
    draw the pixel using the object's color  
}
```



# Z buffer Illustration



Correct Final image



eye

Top View

# Z buffer Illustration



**Step 1:** Initialize the depth buffer

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

Largest possible  
z values is 1.0

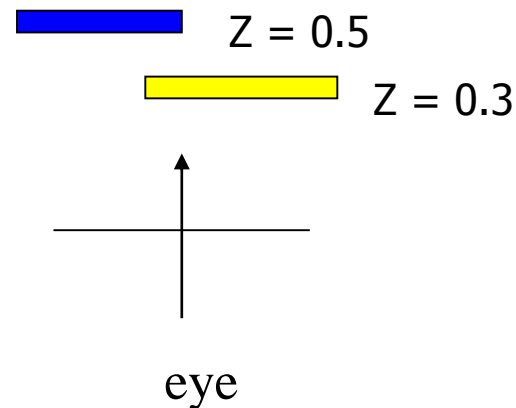




# Z buffer Illustration

**Step 2:** Draw blue polygon  
(actually order does not affect final result)

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
0.5	0.5	1.0	1.0
0.5	0.5	1.0	1.0



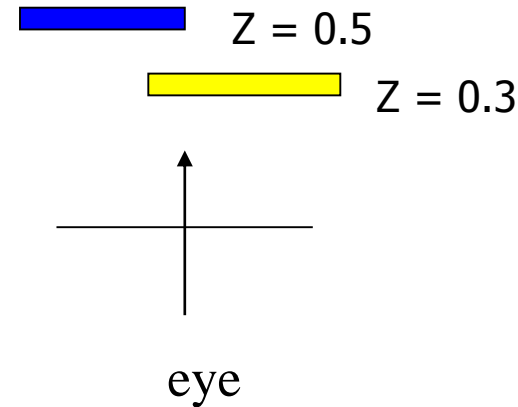
1. Determine group of pixels corresponding to blue polygon
2. Figure out z value of blue polygon for each covered pixel (0.5)
3. For each covered pixel,  $z = 0.5$  is less than 1.0
  1. Smallest z so far = 0.5, color = blue



# Z buffer Illustration

**Step 3:** Draw the yellow polygon

1.0	1.0	1.0	1.0
1.0	0.3	0.3	1.0
0.5	0.3	0.3	1.0
0.5	0.5	1.0	1.0



1. Determine group of pixels corresponding to yellow polygon
2. Figure out z value of yellow polygon for each covered pixel (0.3)
3. For each covered pixel,  $z = 0.3$  becomes minimum, color = yellow

**z-buffer drawback:** wastes resources drawing and redrawing faces



# OpenGL HSR Commands

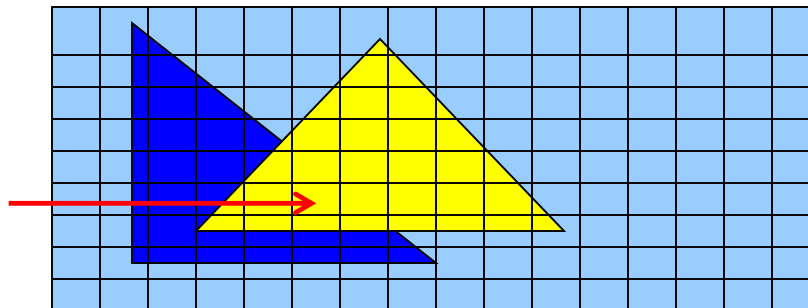
- 3 main commands to do HSR
- `glutInitDisplayMode (GLUT_DEPTH | GLUT_RGB)`  
instructs OpenGL to create depth buffer
- `glEnable (GL_DEPTH_TEST)` enables depth testing
- `glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` initializes depth buffer every time we draw a new picture



# Z-buffer Algorithm

- Initialize every pixel's z value to 1.0
- rasterize every polygon
- For each pixel in polygon, find its z value (interpolate)
- Track smallest z value so far through each pixel
- As we rasterize polygon, for each pixel in polygon
  - If polygon's z through this pixel < current min z through pixel
  - Paint pixel with polygon's color

**Find depth (z) of every polygon at each pixel**





# Z (depth) Buffer Algorithm

Depth of polygon being rasterized at pixel (x, y)

Largest depth seen so far Through pixel (x, y)

```
For each polygon {  
  for each pixel (x,y) in polygon area {  
    if (z_polygon_pixel(x,y) < depth_buffer(x,y) ) {  
      depth_buffer(x,y) = z_polygon_pixel(x,y);  
      color_buffer(x,y) = polygon color at (x,y)  
    }  
  }  
}
```

**Note: know depths at vertices. Interpolate for interior z\_polygon\_pixel(x, y) depths**

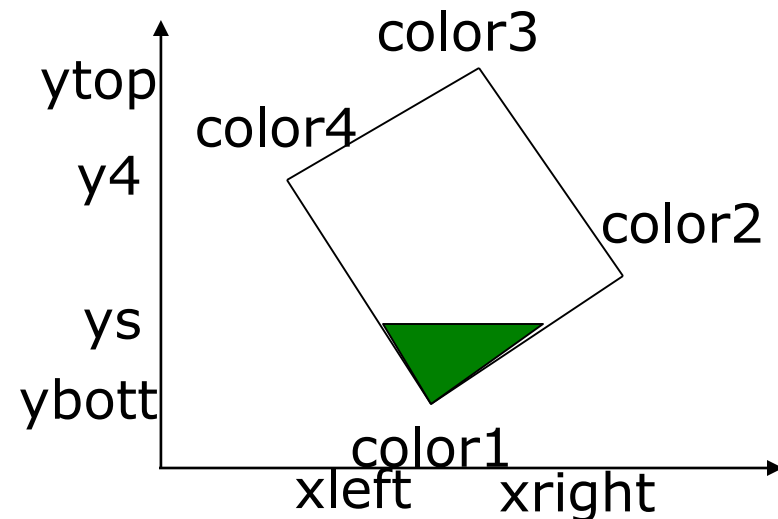
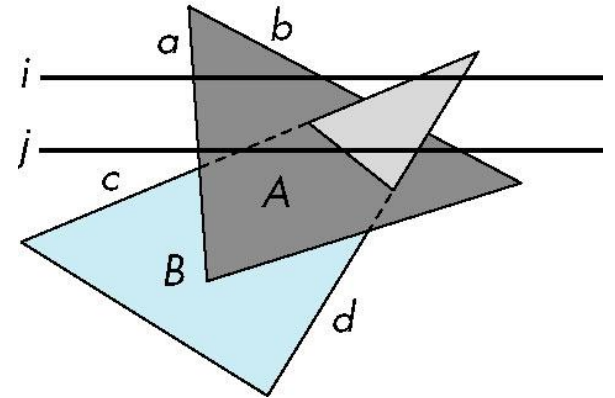




# Combined z-buffer and Gouraud Shading (Hill)

- Can combine shading and hsr through scan line algorithm

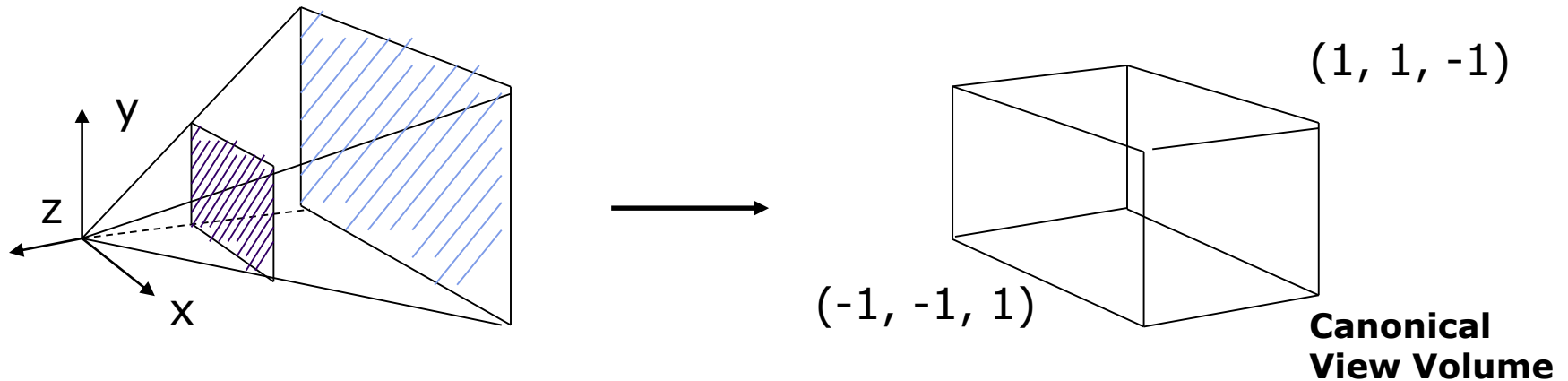
```
for(int y = ybott; y <= ytop; y++) // for each scan line
{
    for(each polygon){
        find xleft and xright
        find dleft, dright, and dinc
        find colorleft and colorright, and colorinc
        for(int x = xleft, c = colorleft, d = dleft; x <= xright;
            x++, c+= colorinc, d+= dinc)
        {
            if(d < d[x][y])
            {
                put c into the pixel at (x, y)
                d[x][y] = d; // update closest depth
            }
        }
    }
}
```



# Perspective Transformation: Z-Buffer Depth Compression



- **Pseudodepth calculation:** Recall we chose parameters (a and b) to map z from range [near, far] to **pseudodepth** range[-1,1]



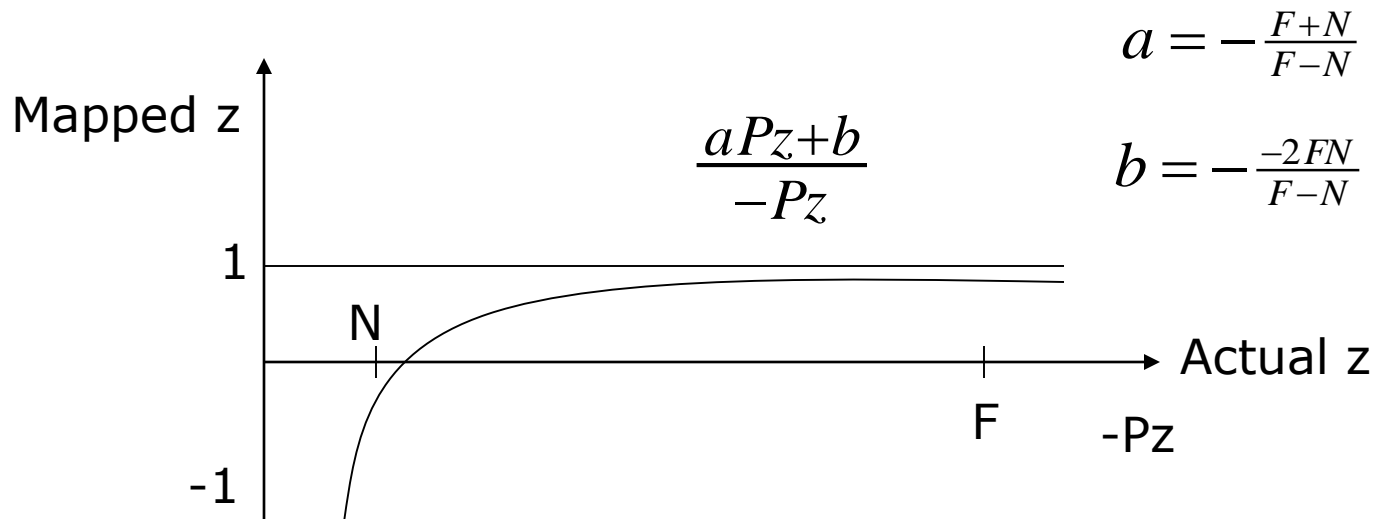
$$\begin{pmatrix} \frac{2N}{x_{\max} - x_{\min}} & 0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0 \\ 0 & \frac{2N}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0 \\ 0 & 0 & \frac{-(F + N)}{F - N} & \frac{-2FN}{F - N} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

**These values map z values of original view volume to [-1, 1] range**



# Z-Buffer Depth Compression

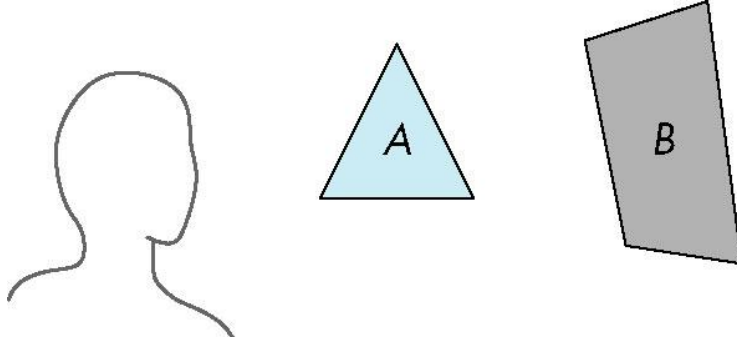
- This mapping is almost linear close to eye
- Non-linear further from eye, approaches asymptote
- Also limited number of bits
- Thus, two z values close to far plane may map to same pseudodepth: **Errors!!**



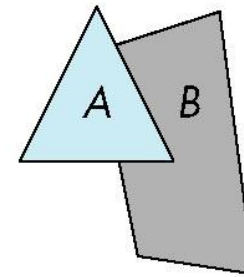


# Painter's HSR Algorithm

- Render polygons farthest to nearest
- Similar to painter layers oil paint



Viewer sees B behind A



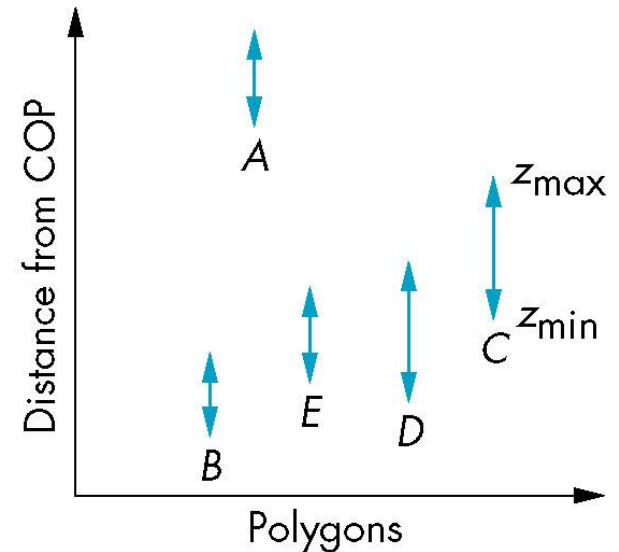
Render B then A



# Depth Sort

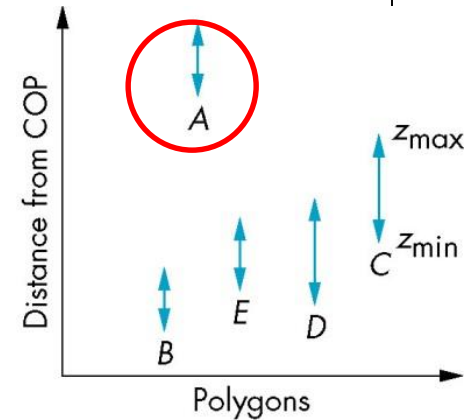
- Requires sorting polygons (based on depth)
  - $O(n \log n)$  complexity to sort  $n$  polygon depths
  - Not every polygon is clearly in front or behind other polygons

Polygons sorted by distance from COP

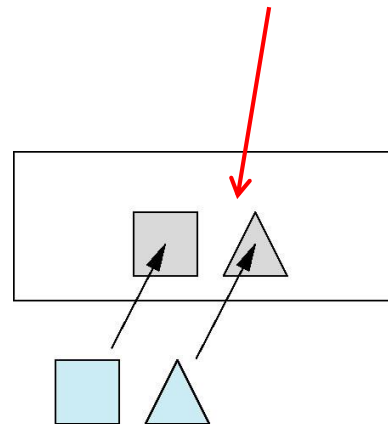


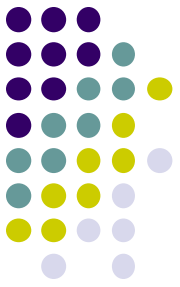
# Easy Cases

- Case a: A lies behind all polygons

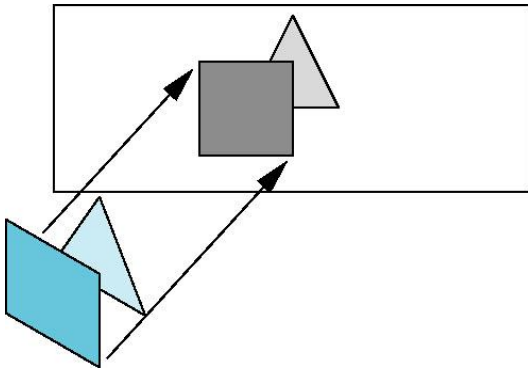


- Case b: Polygons overlap in z but **not** in x or y

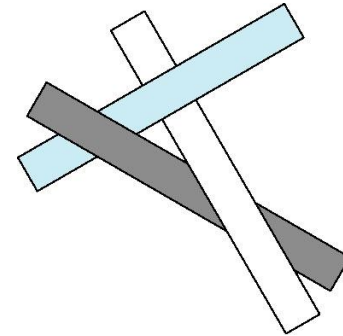




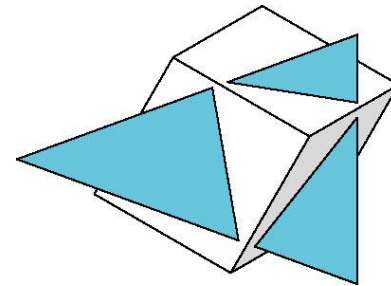
# Hard Cases



Overlap in (x,y) and z ranges



cyclic overlap

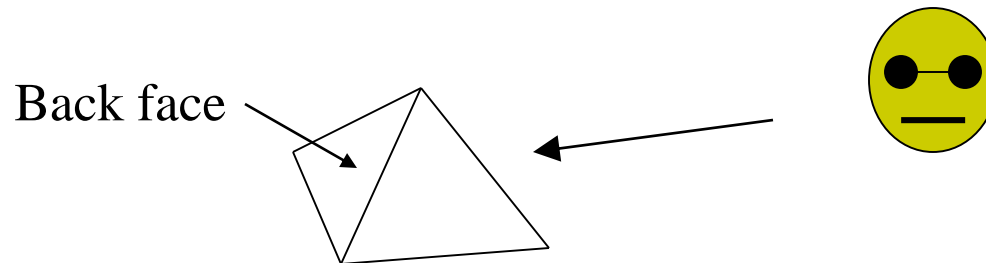


penetration



# Back Face Culling

- **Back faces:** faces of opaque object that are “pointing away” from viewer
- **Back face culling:** do not draw back faces (saves resources)



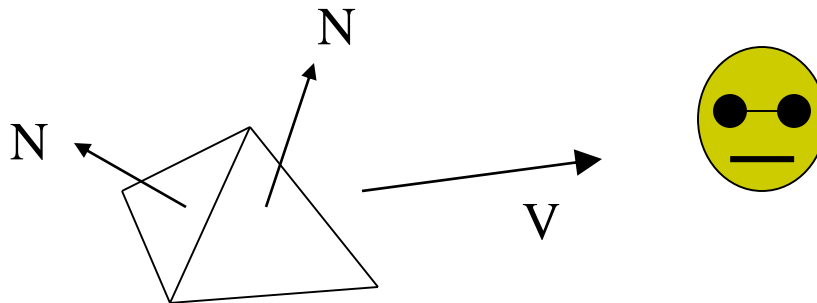
- How to detect back faces?





# Back Face Culling

- Goal: Test if a face F is is backface
- How? Form vectors
  - View vector,  $V$
  - Normal  $N$  to face  $F$



**Backface test: F is backface if  $N \cdot V < 0$  why??**



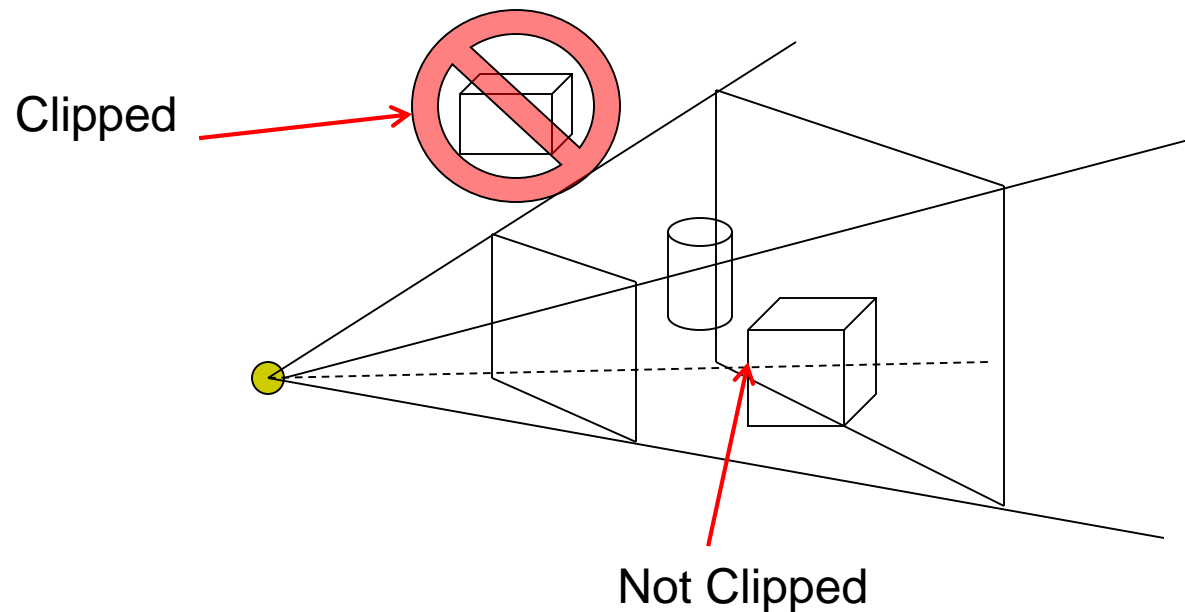
# Back Face Culling: Draw mesh front faces

```
void drawFrontFaces( )  
{  
    for(int f = 0;f < numFaces; f++)  
    {  
        if(isBackFace(f, ....) continue; ← if N.V < 0  
        glDrawArrays(GL_POLYGON, 0, N);  
    }  
}
```



# View-Frustum Culling

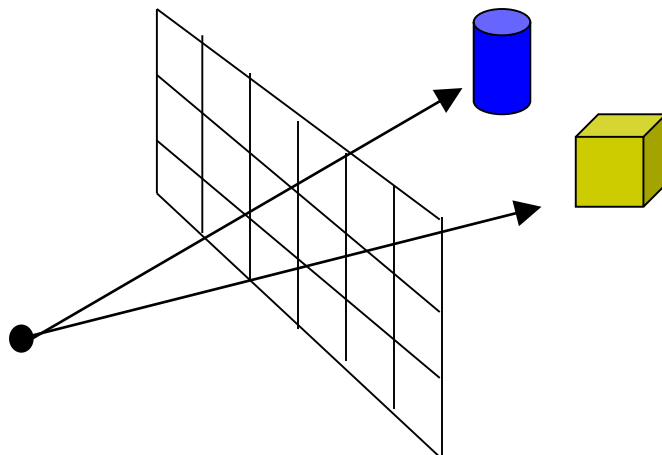
- **Goal:** Remove objects outside view frustum
- Done by 3D clipping algorithm (e.g. Liang-Barsky)





# Ray Tracing

- Ray tracing is another image space method
- Ray tracing: Cast a ray from eye through each pixel into world.
- Ray tracing algorithm figures out: what object seen in direction through given pixel?



Overview later



# References

- Angel and Shreiner, Interactive Computer Graphics, 6<sup>th</sup> edition
- Hill and Kelley, Computer Graphics using OpenGL, 3<sup>rd</sup> edition, Chapter 9