ME/CprE/ComS 557

# Computer Graphics and Geometric Modeling

## Rendering

October 1st, 2015

Rafael Radkowski

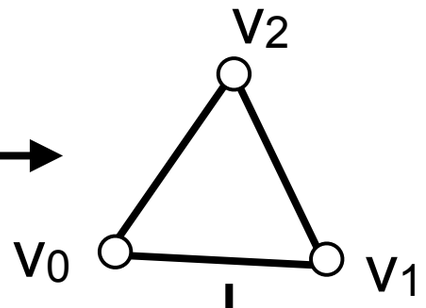
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Content

ARLAB

- Rendering algorithm and rasterization
- Shading / Fragment Shader
- Depth Test

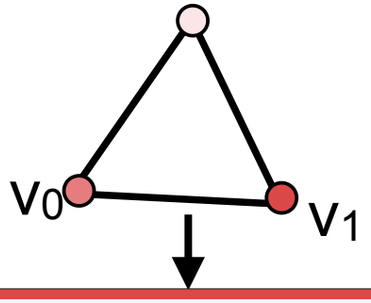
# Rendering: from model to pixel




***Display List*** *of a 3D model*


- Calculate vertex colors
- Primitive Assembly

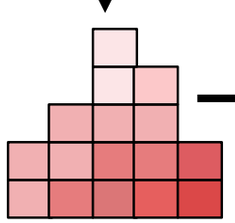
Each primitive (point, line, polygon, quad etc.) is processed individually.





- Shading: fills the primitive (e.g., polygon, quad, etc.) with color

*Image data in* ***Frame Buffer***

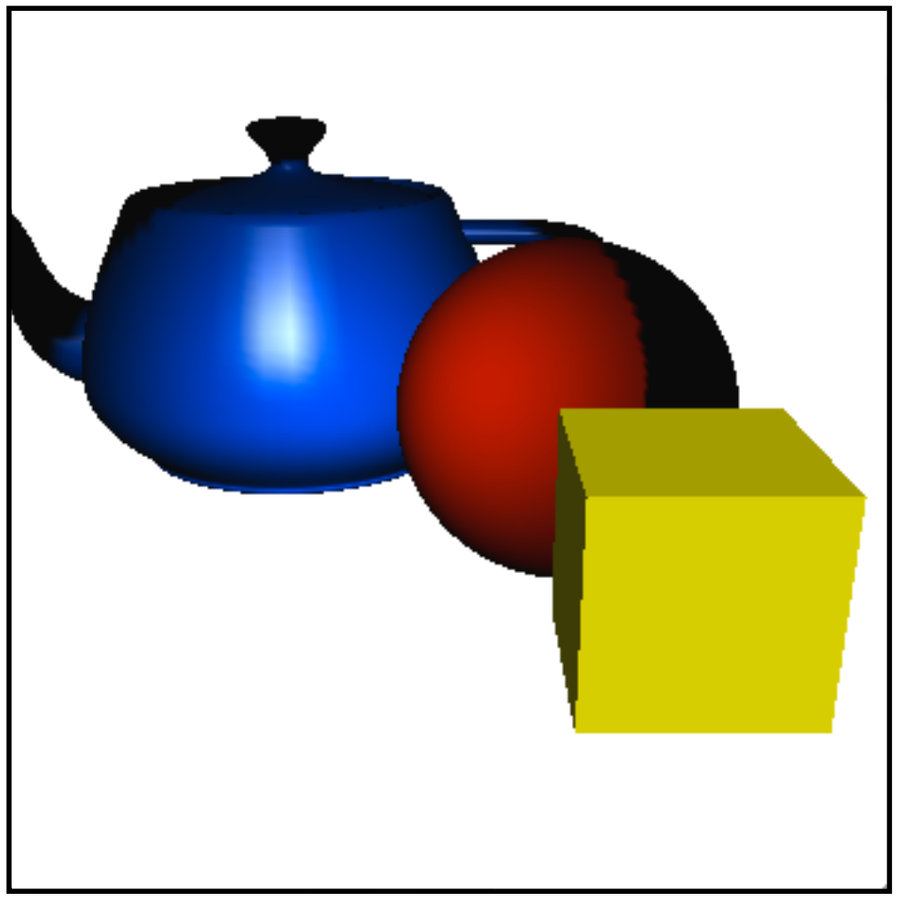


VRAC | HCI

# Rendering Algorithm / Rasterization

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Sequence of Drawing (1/2)



*Output on display*

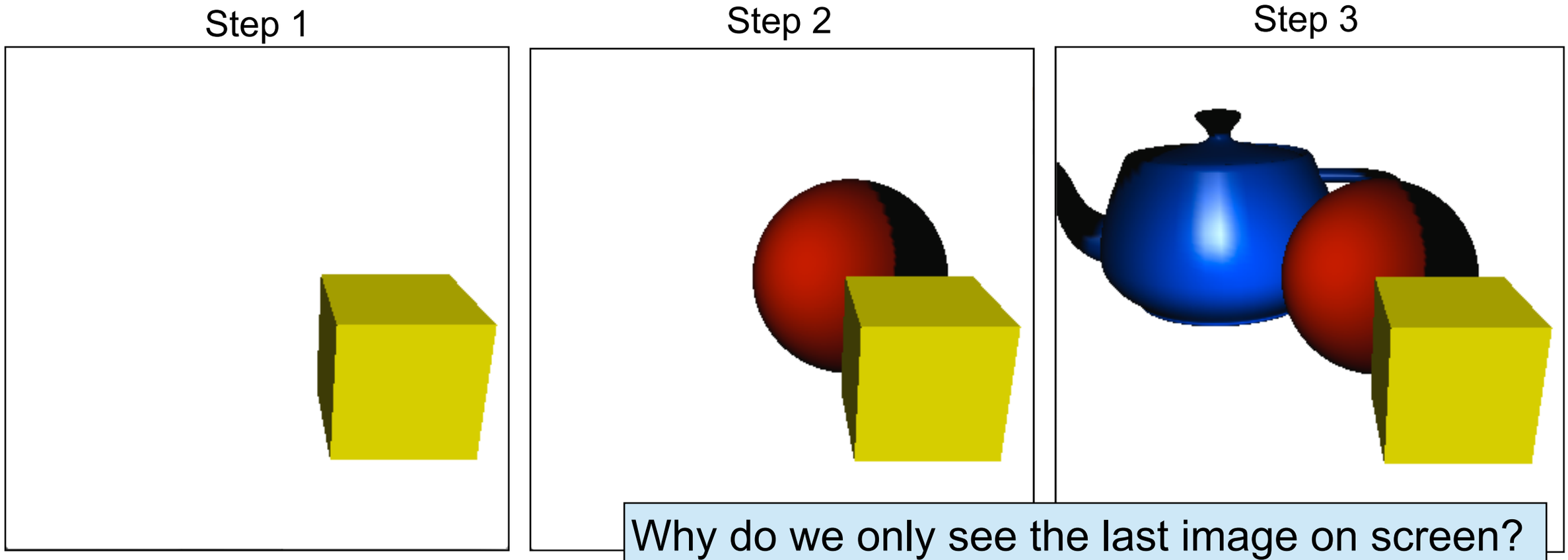
**Code snippet of the display function:**

(Every function encases all function calls that are necessary to draw the objects)

```
void draw_scene(void)
{
    // draw solid cube
    draw_solid_cube();

    // draw a solid sphere
    draw_solid_sphere();

    // draw a teapot
    draw_solid_teapot();
}
```

What is the sequence of drawing?

# Sequence of Drawing (2/2)




*Output on display*

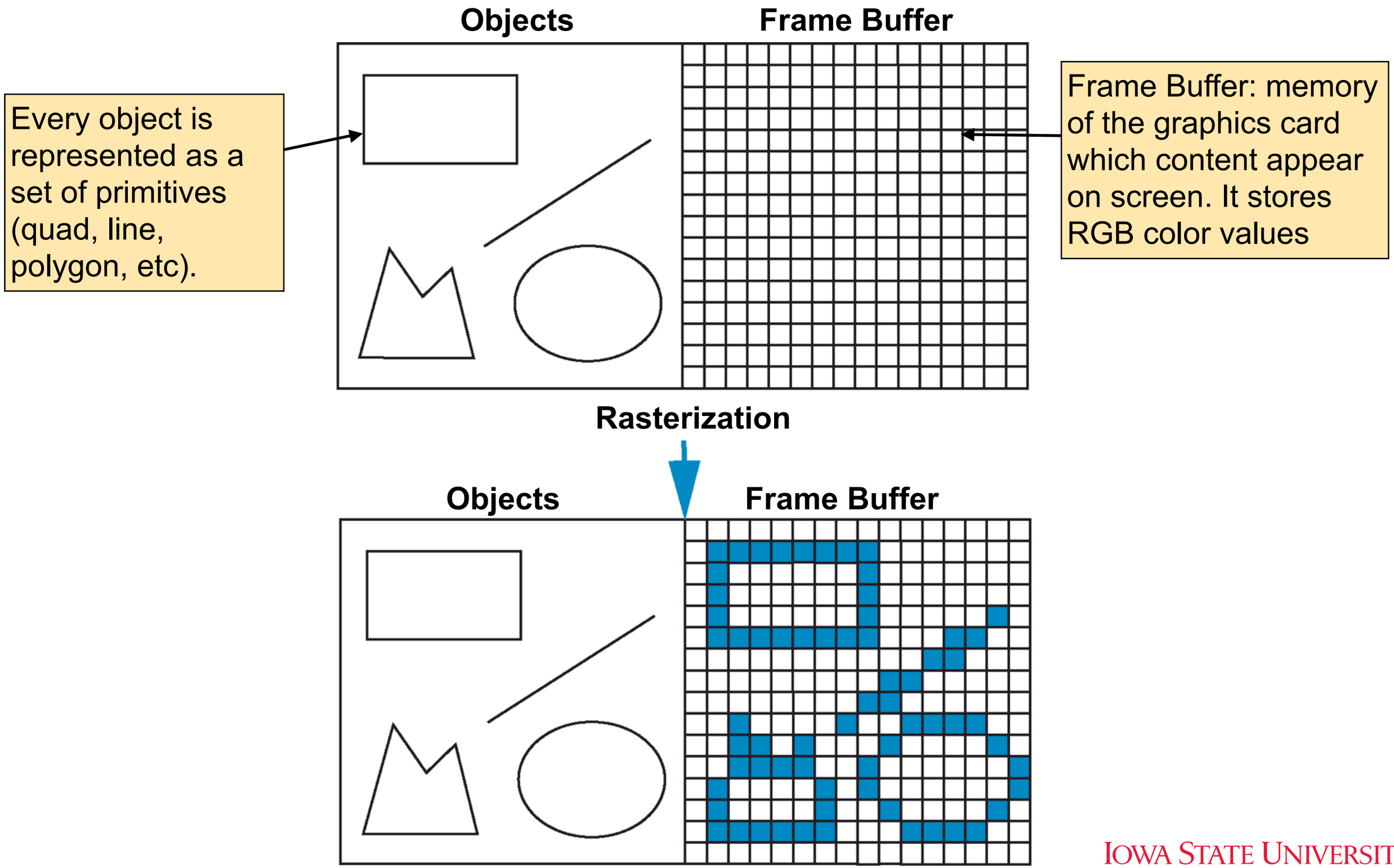
Why do we only see the last image on screen?

*Sequence of execution*

```
// draw solid cube
draw_solid_cube();

// draw a solid sphere
draw_solid_sphere();

// draw a teapot
draw_solid_teapot();
```

Code is executed from the top to the bottom of the file. The rendering algorithm draws the object in the sequence on screen in which the objects appear in the code.

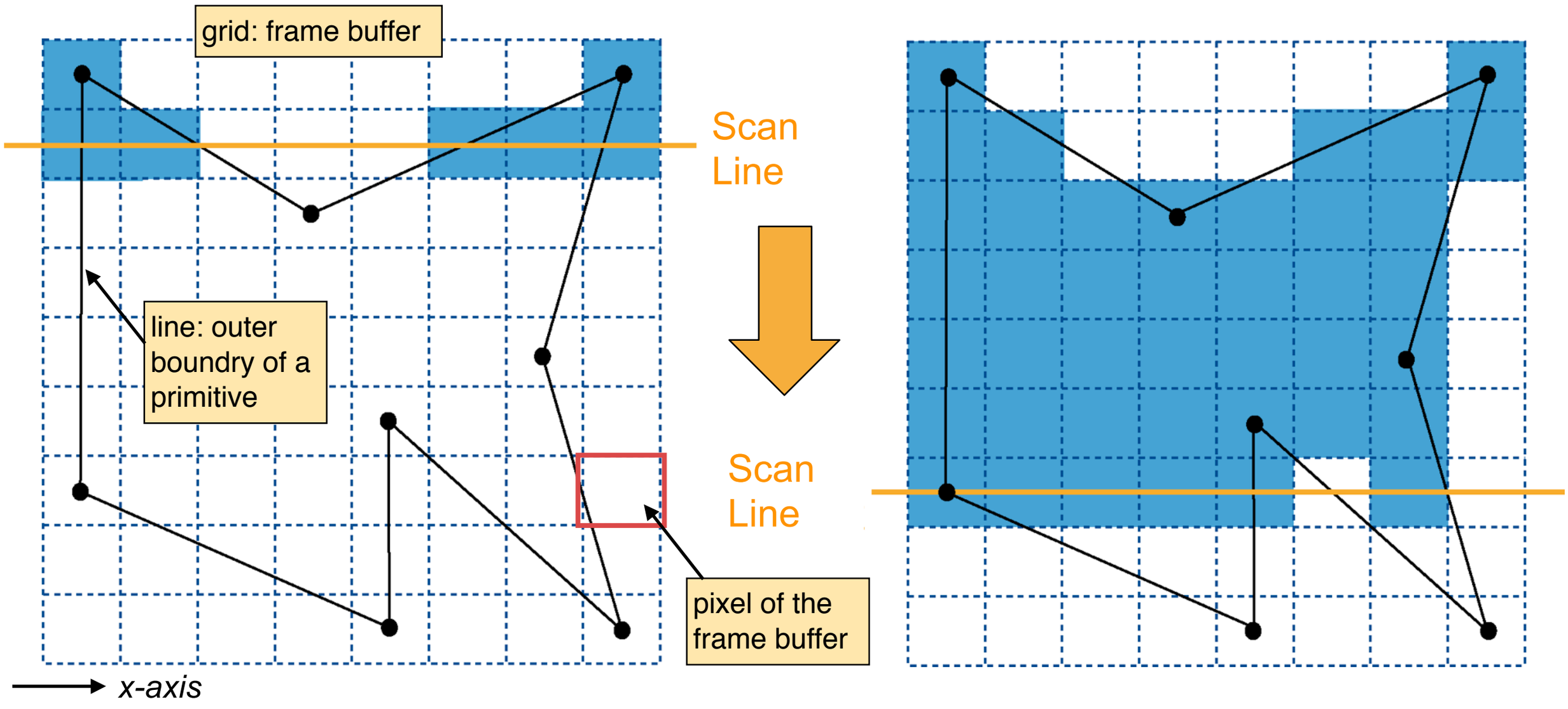
# Rasterization



# Filling of Surfaces -
*Scan-Line-Algorithm*

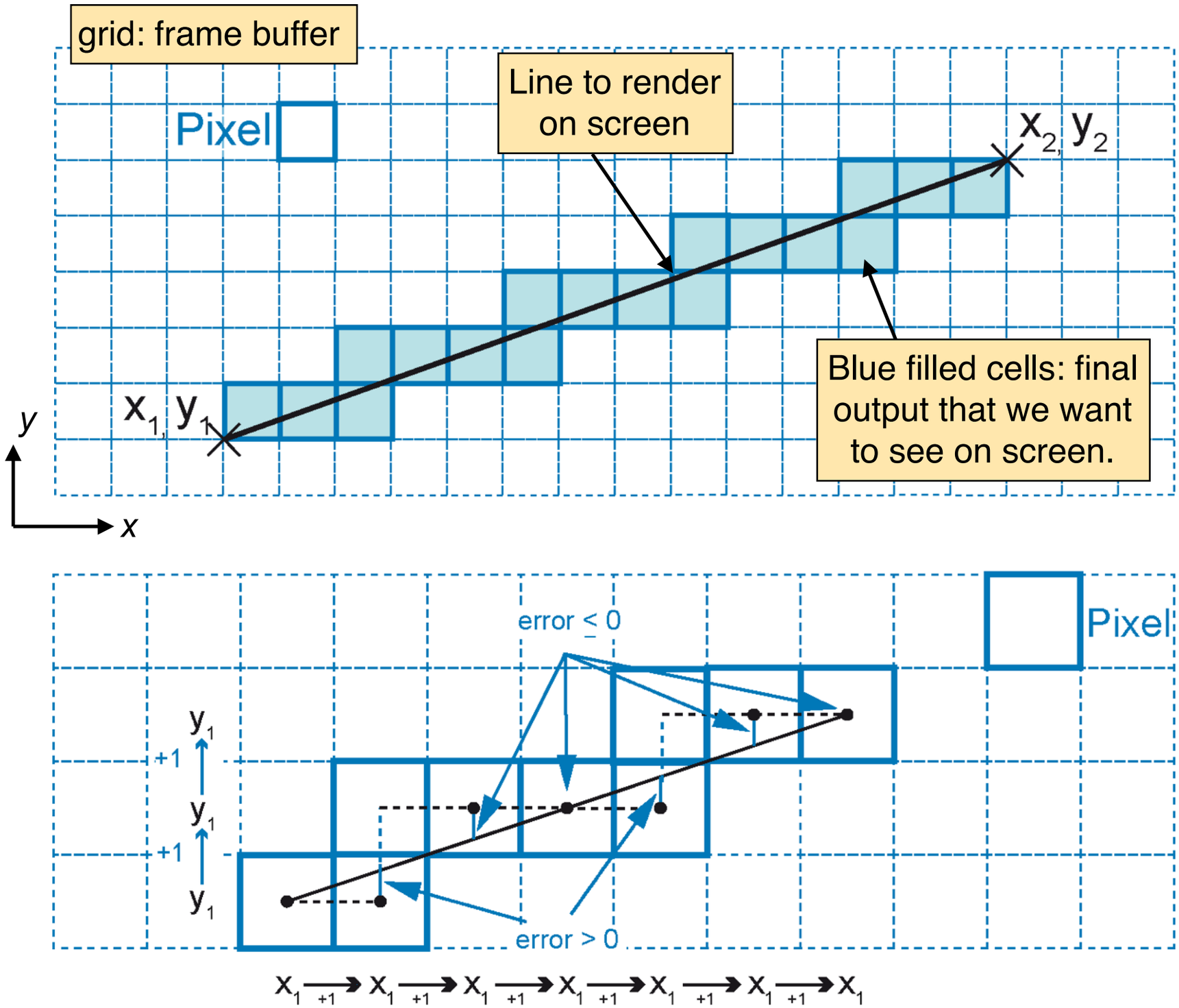
VRAC | HCI

The Scan-Line-Algorithm is a method to fill surfaces of OpenGL primitives




The **Scan-Line-Algorithm** fills a primitive by pushing a scan line in parallel to the x-axis from the top of the screen to the very bottom. The intersection of the scan line with the outer boundry are calculated and all pixels within the primitive are filled (switched on; at this time, no color has been calculated). The algorithm only works with closed primitives.

# Rasterization of Lines -
*Bresenham-Algorithm*




The goal of the rasterization of a line is to select a set of pixels that are close to the line which shall be rendered on screen.

The Bresenham-Algorithm (1965) is an efficient way to render lines. It allows to raster a line by using only additions and subtraction operations.

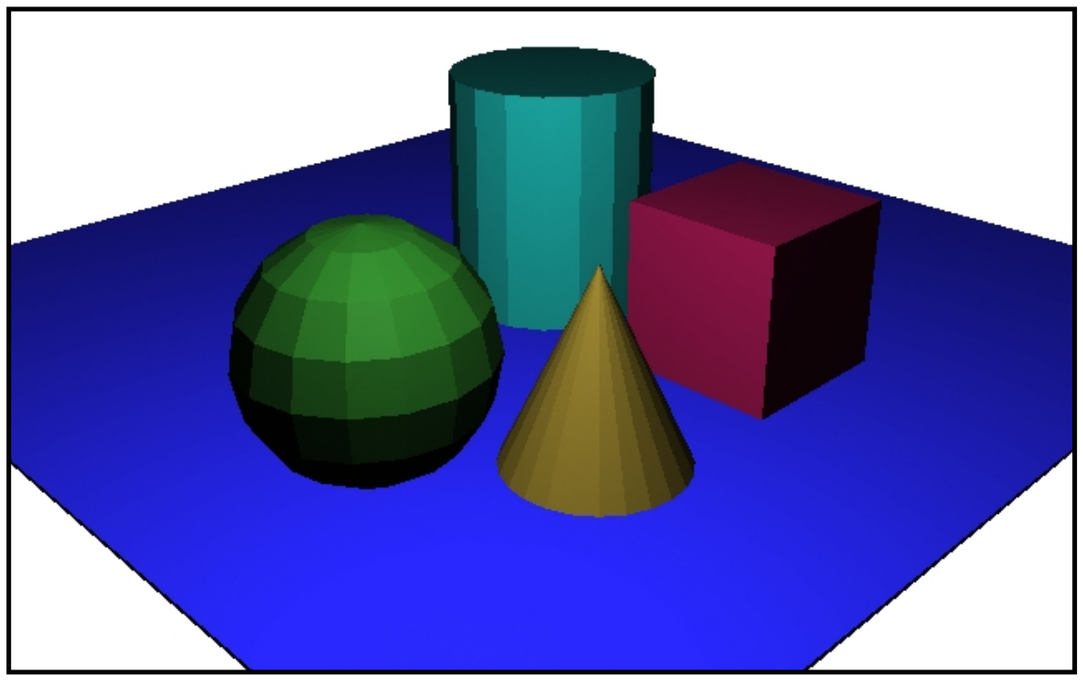
- Start point for the algorithm is the start of a line / line loop. The start pixel is drawn on screen.
- Every step, the x-value is increased by one.
- The error on y between the ideal line and the center of the pixel is calculated.
- y is increased by one if the error rate exceeds a threshold.

VRAC | HCI

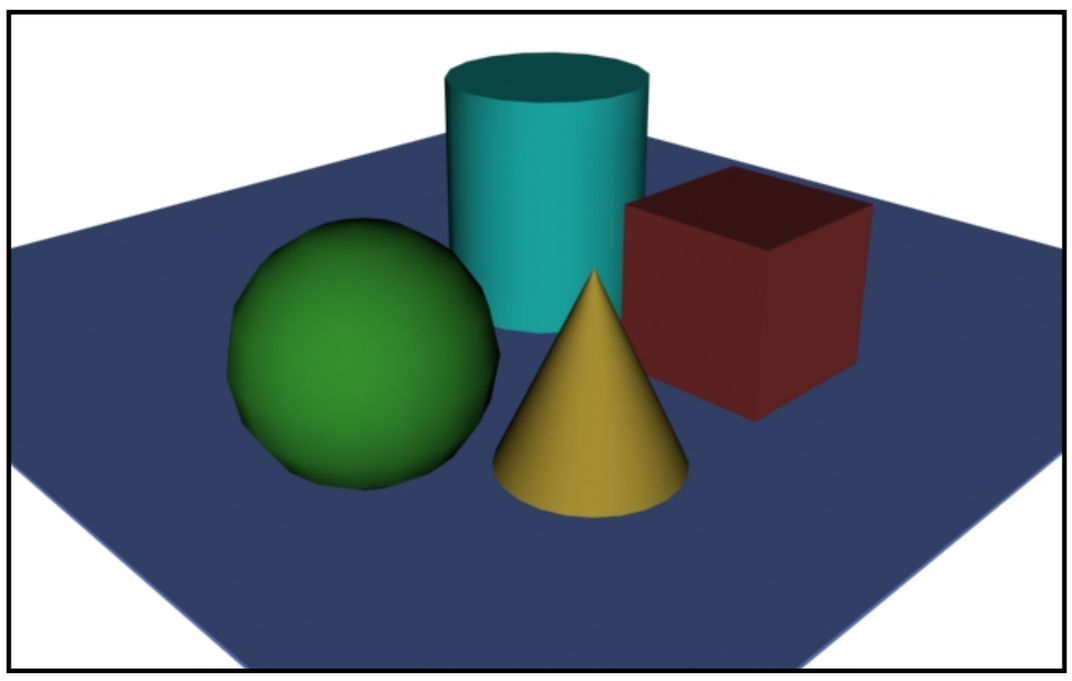
# Shading / Fragment Shader

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

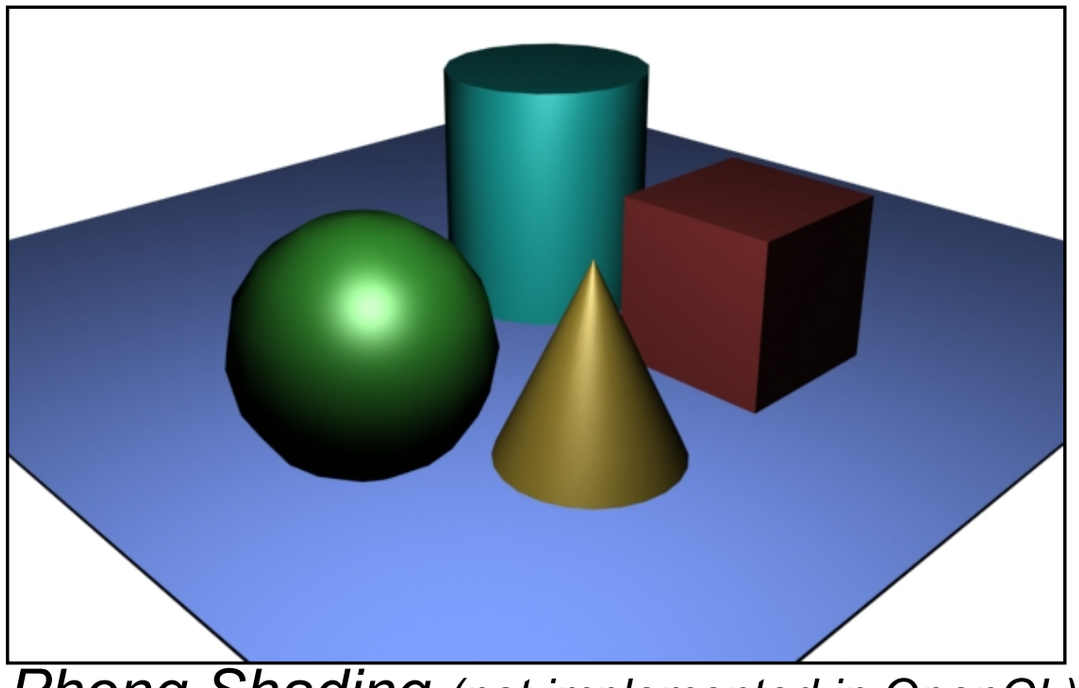
# Shading



*Flat Shading*



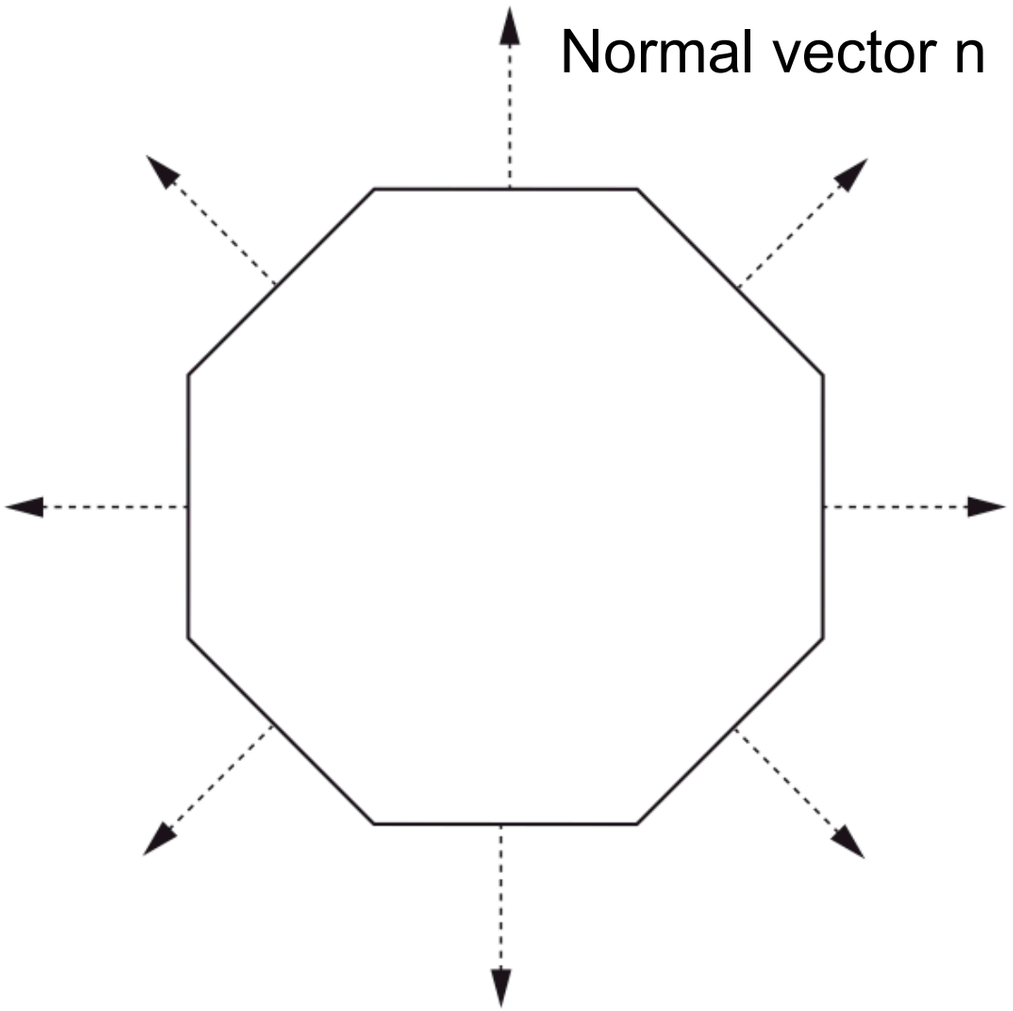
*Gouraud Shading*

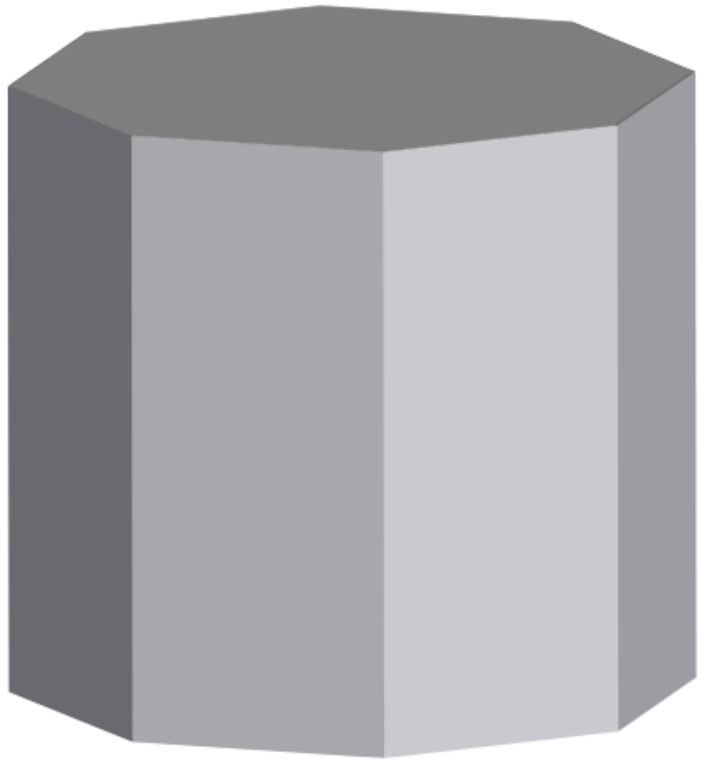


*Phong Shading (not implemented in OpenGL)*

Shading is the process that fills the primitives (e.g., polygons, triangles, quads etc.) with color. The task is to determine a color value for each pixel on screen.

# Flat Shading

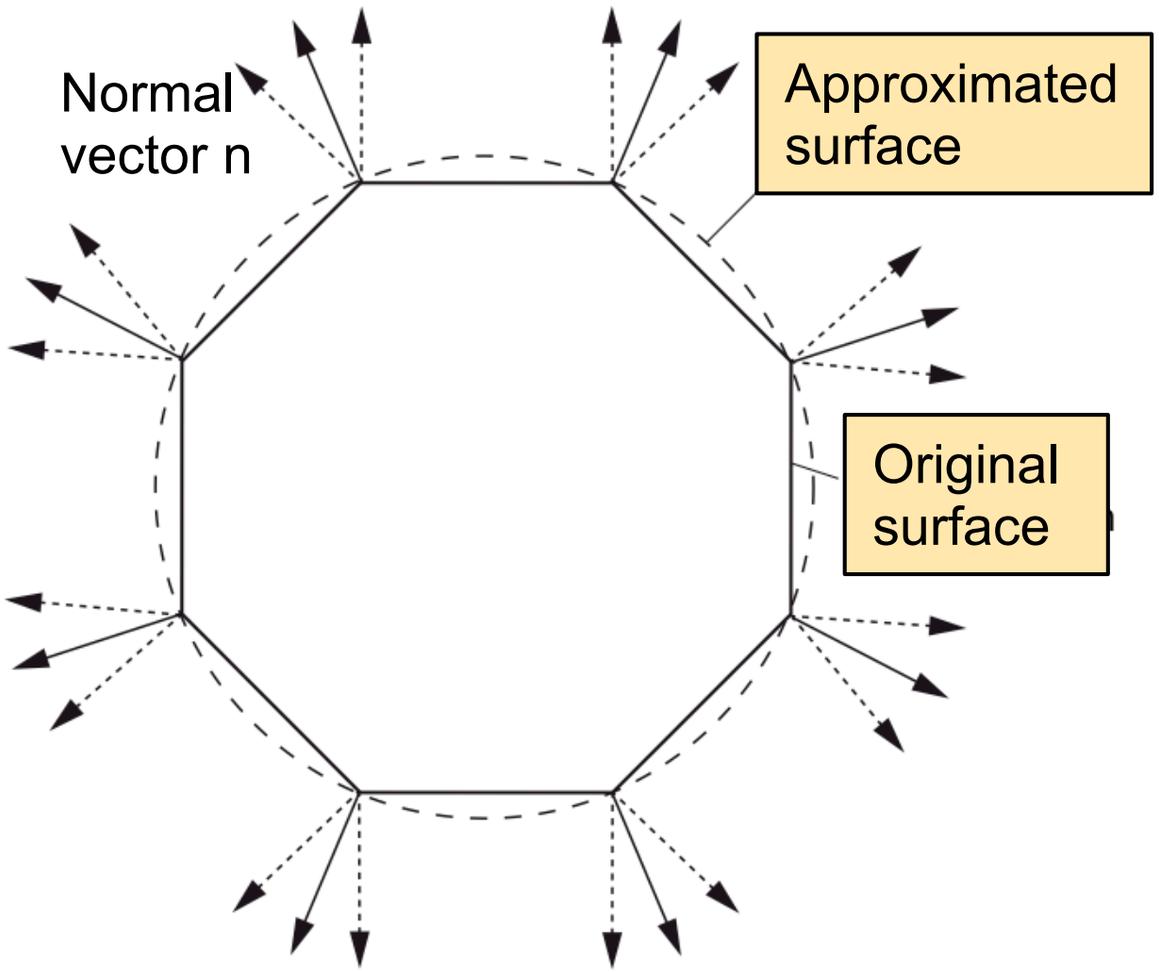


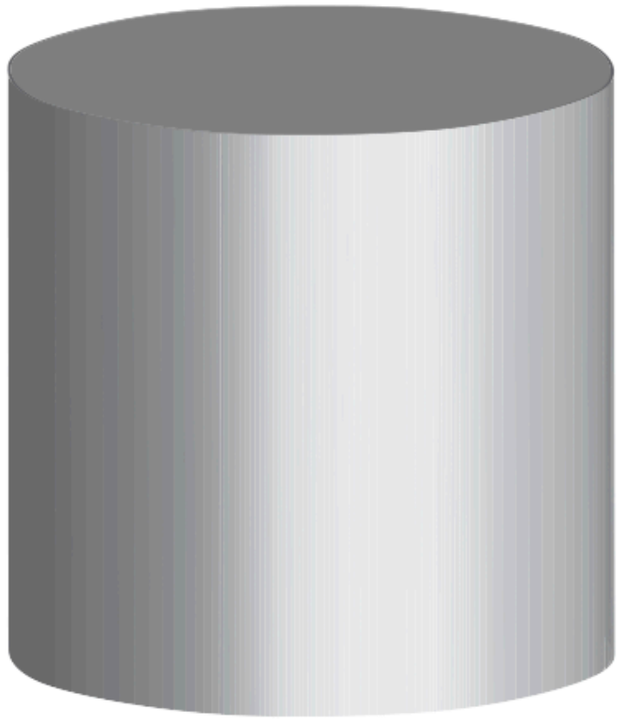



*The Flat Shading method calculates one color for each surface with respect to the surface normal.*

- Accentuates the individual polygons
- One color is assigned to each polygon
- Very fast
- Color is computed for one vertex and assigned to entire polygon
- Specular highlights are rendered poorly

# Gouraud Shading (1/2)

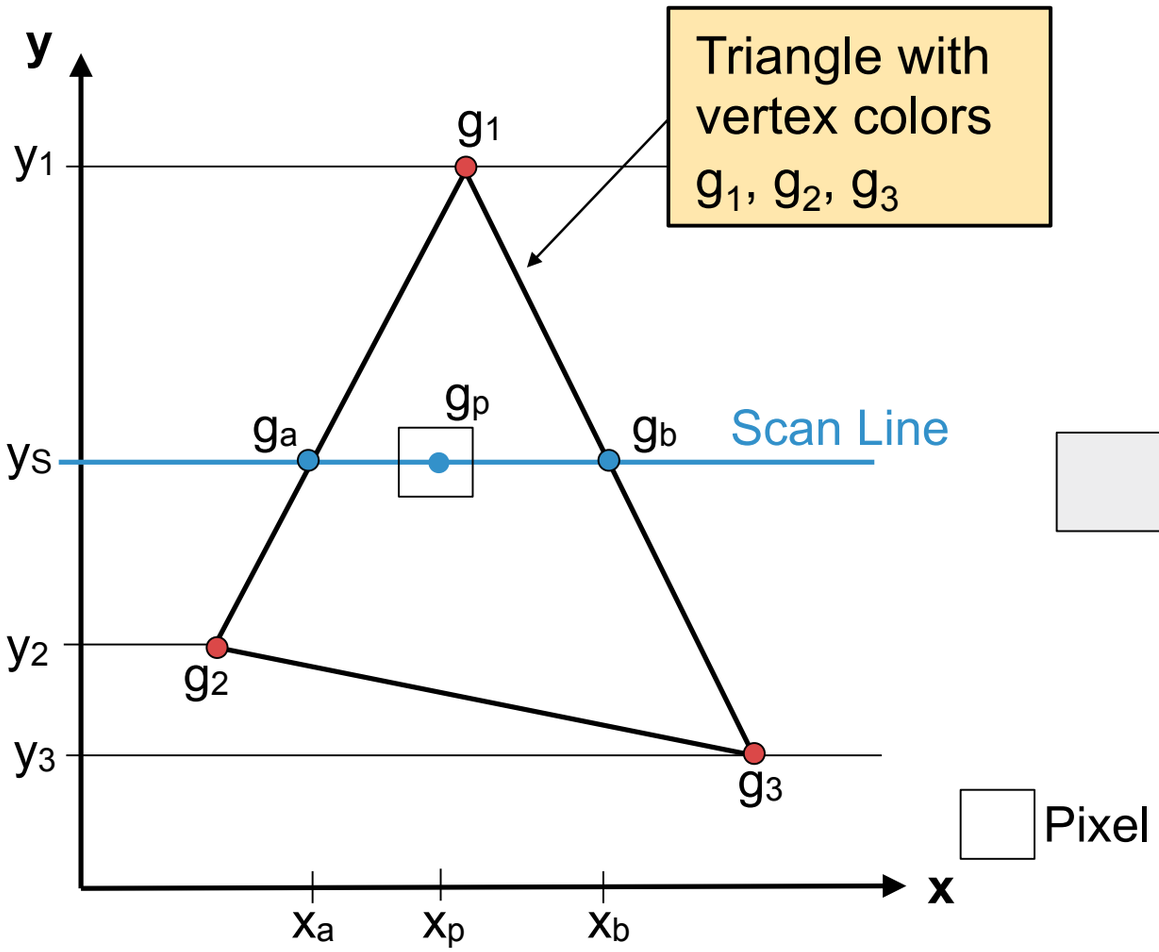


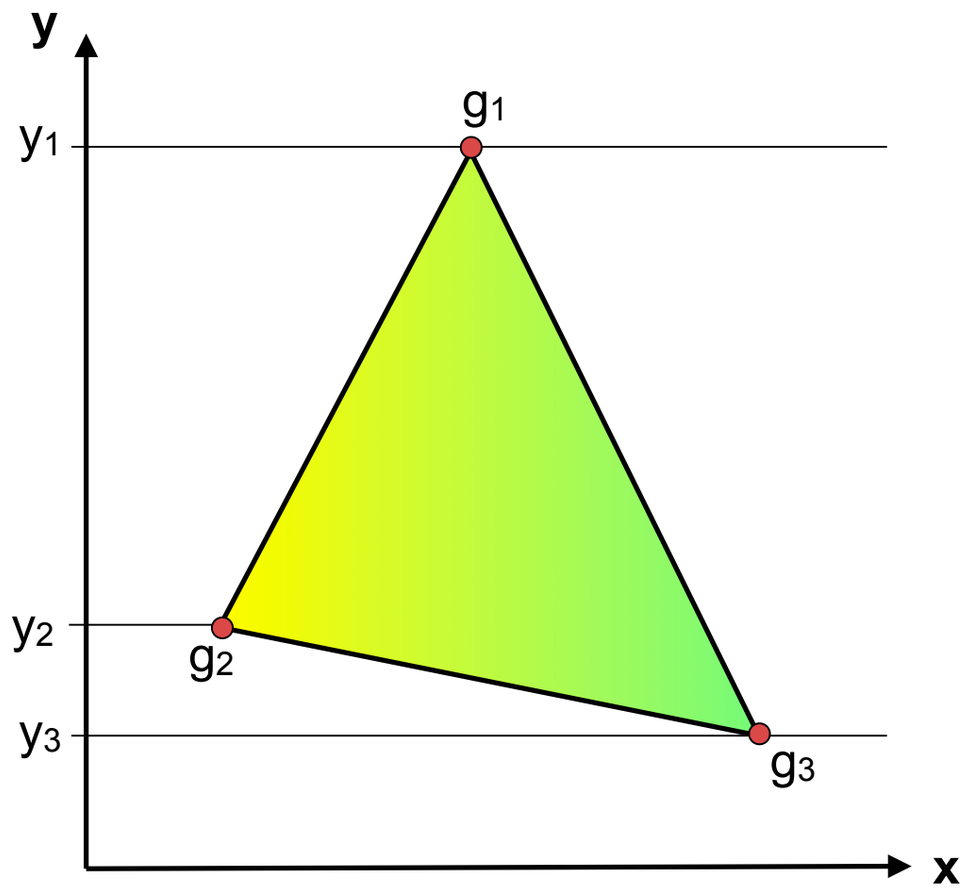



*Calculate the normal vectors at each contact vertex between two surfaces in order to approximate a smooth surface.*

- Smooths the edges between polygons
- Color is linearly interpolated from vertex points
- Specular highlights appear interpolated

# Gouraud Shading (2/2)




The interpolation of a pixel color takes two steps: first, the color at the triangle edges are calculated. Secondly, the pixel color is calculated.

$$g_a = g_1 - (g_1 - g_2) \cdot \frac{y_1 - y_s}{y_1 - y_2}$$

$$g_b = g_1 - (g_1 - g_3) \cdot \frac{y_1 - y_s}{y_1 - y_3}$$

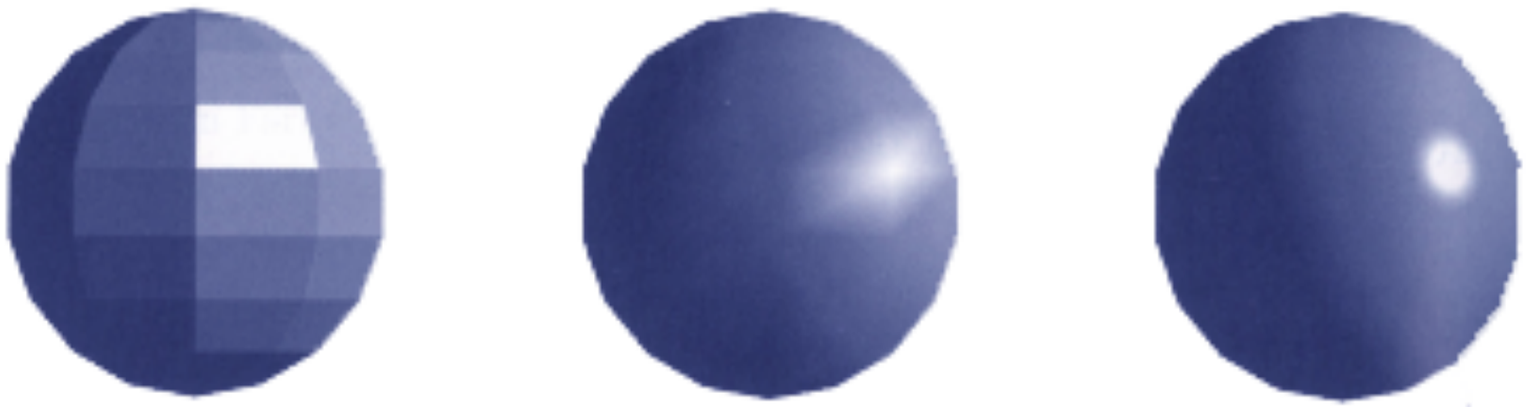
$$g_p = g_b - (g_b - g_a) \cdot \frac{x_b - x_p}{x_b - x_a}$$

$g_{a,\,b}$: Color at the polygon edge
$g_p$: Color of the pixel

# Comparison

VRAC | HCI

**Flat Shading** **Gouraud Shading** **Phong Shading**



**81 Polygons**

**2500 Polygons**

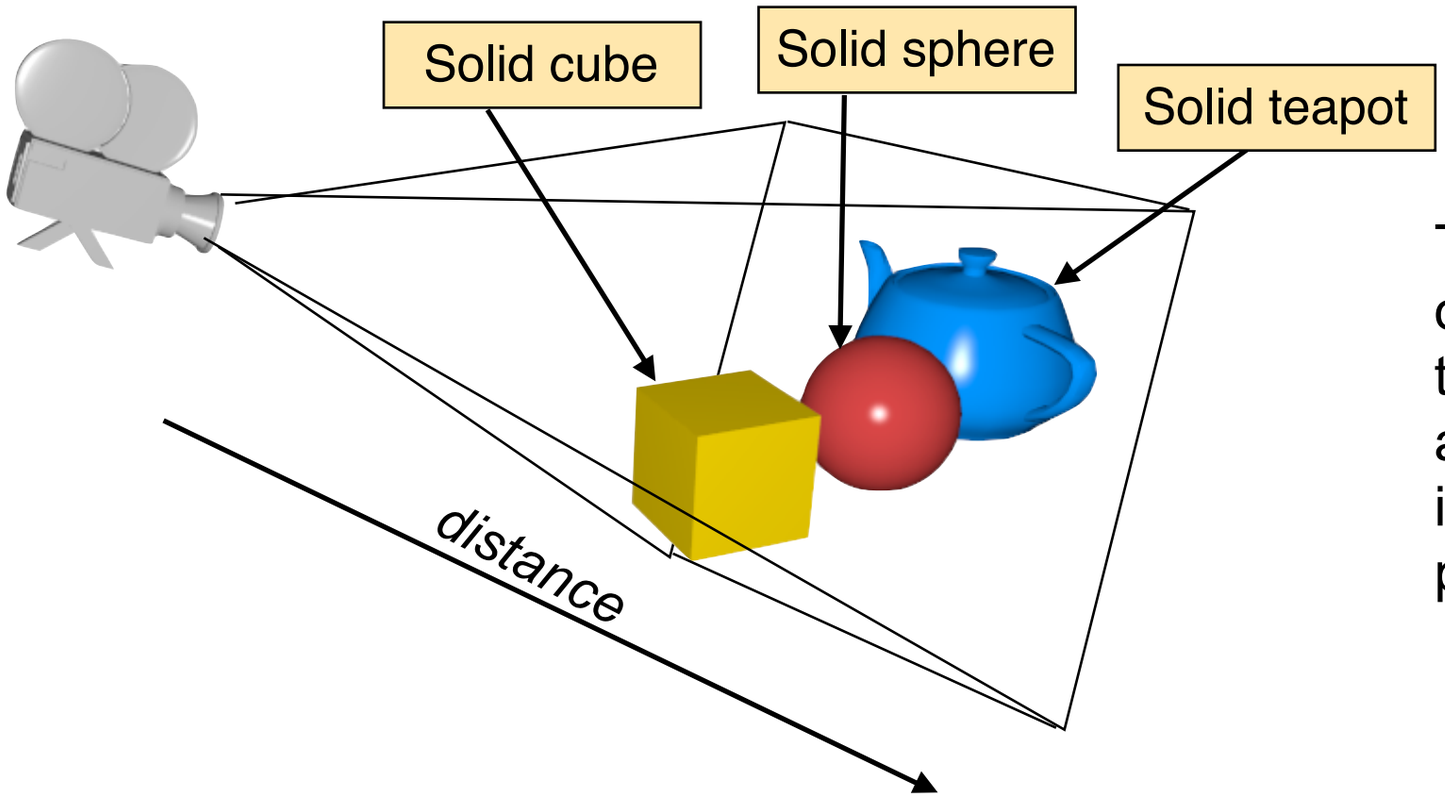
**25000 Polygons**

VRAC | HCI

# Depth Test

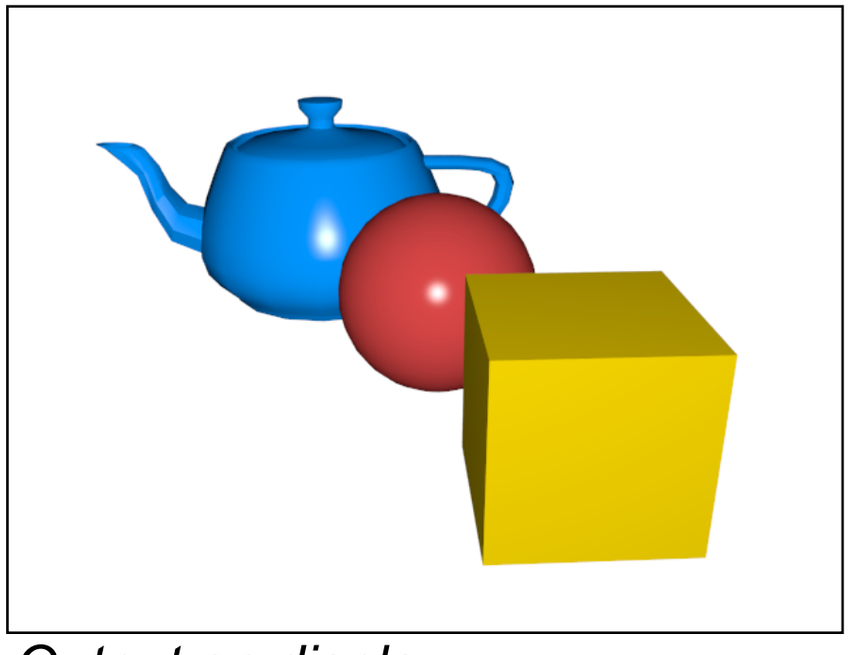
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Considering the Render Sequence




The rendering sequence of all objects depends on their location in the code. The object that primitves are rendered first are also written into the frame buffer in the very first place.

```
void draw_scene(void)
{
    // draw solid cube
    draw_solid_cube();

    // draw a solid sphere
    draw_solid_sphere();

    // draw a teapot
    draw_solid_teapot();
}
```



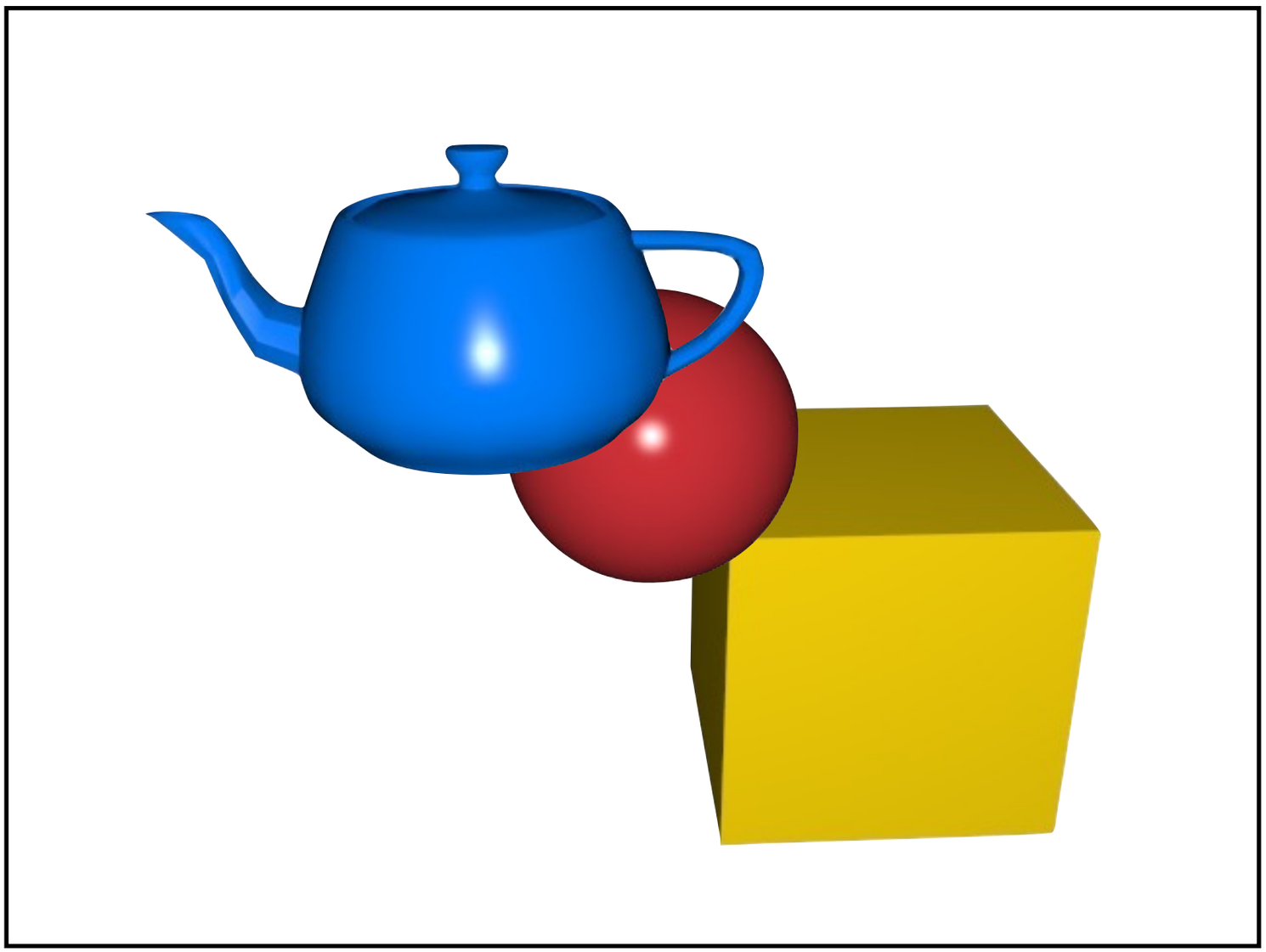
*Output on display*

# Considering the Render Sequence

VRAC|HCI

**Code snippet of the display function:**

```
void draw_scene(void)
{

   // draw solid cube
   draw_solid_cube();

   // draw a solid sphere
   draw_solid_sphere();

   // draw a teapot
   draw_solid_teapot();

}
```



*Output on display*

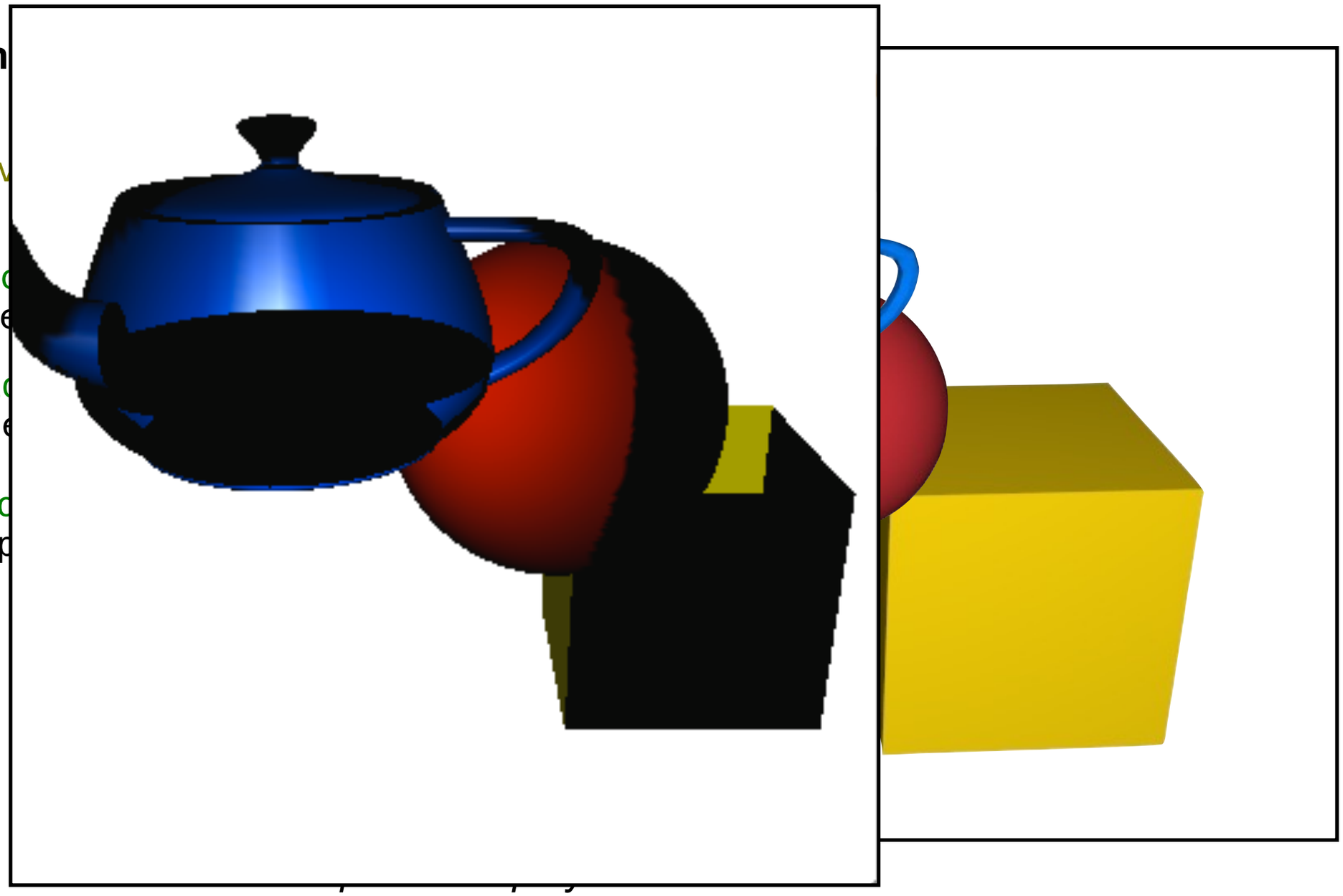
Why do we not see something like this?

# Considering the Render Sequence

VRAC|HCI

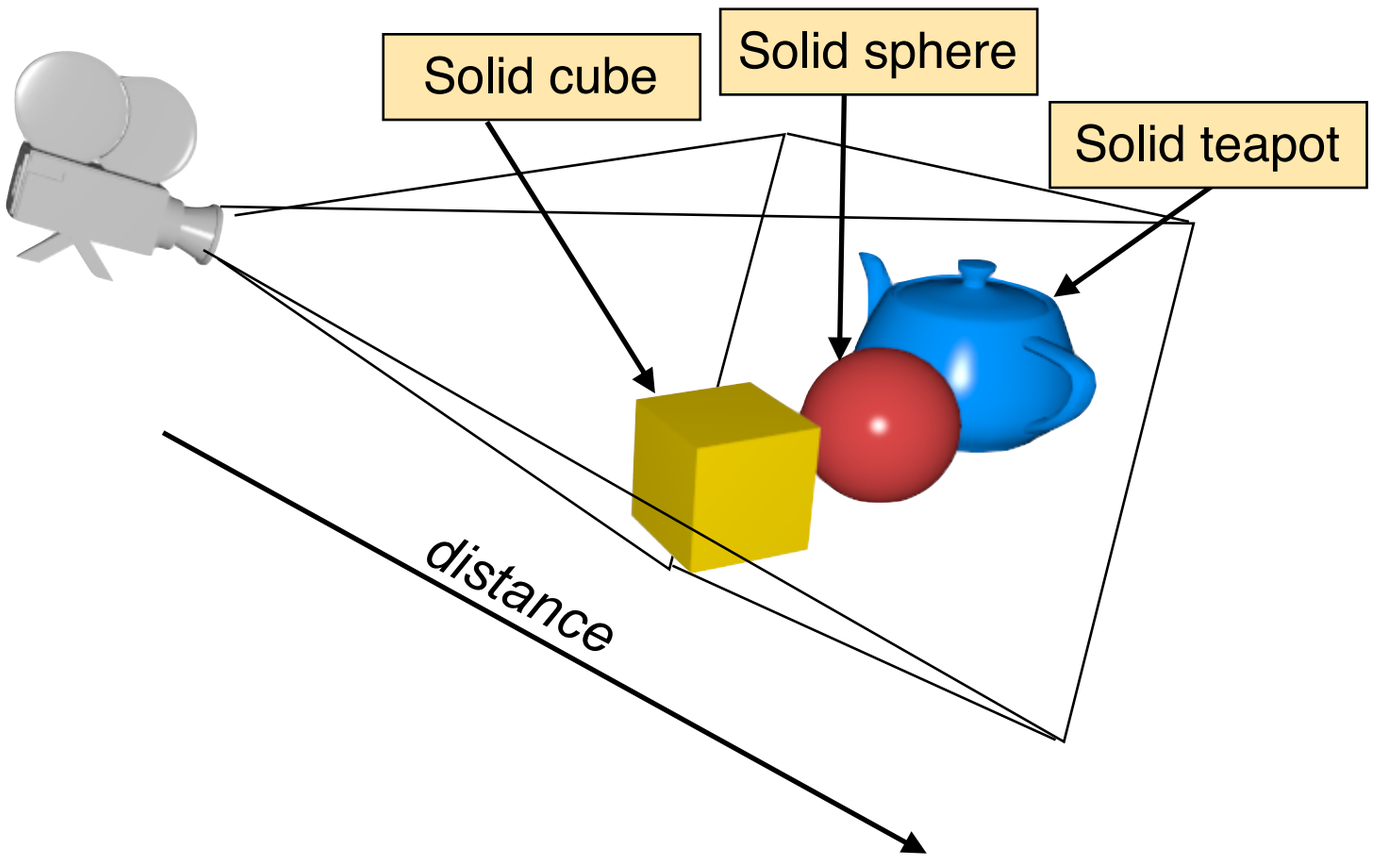
**Code snippet of th**
**display function:**

```
void draw_scene(v
{

   // draw solid c
   draw_solid_cube

   // draw a solic
   draw_solid_sphe

   // draw a teapc
   draw_solid_teap
}
```

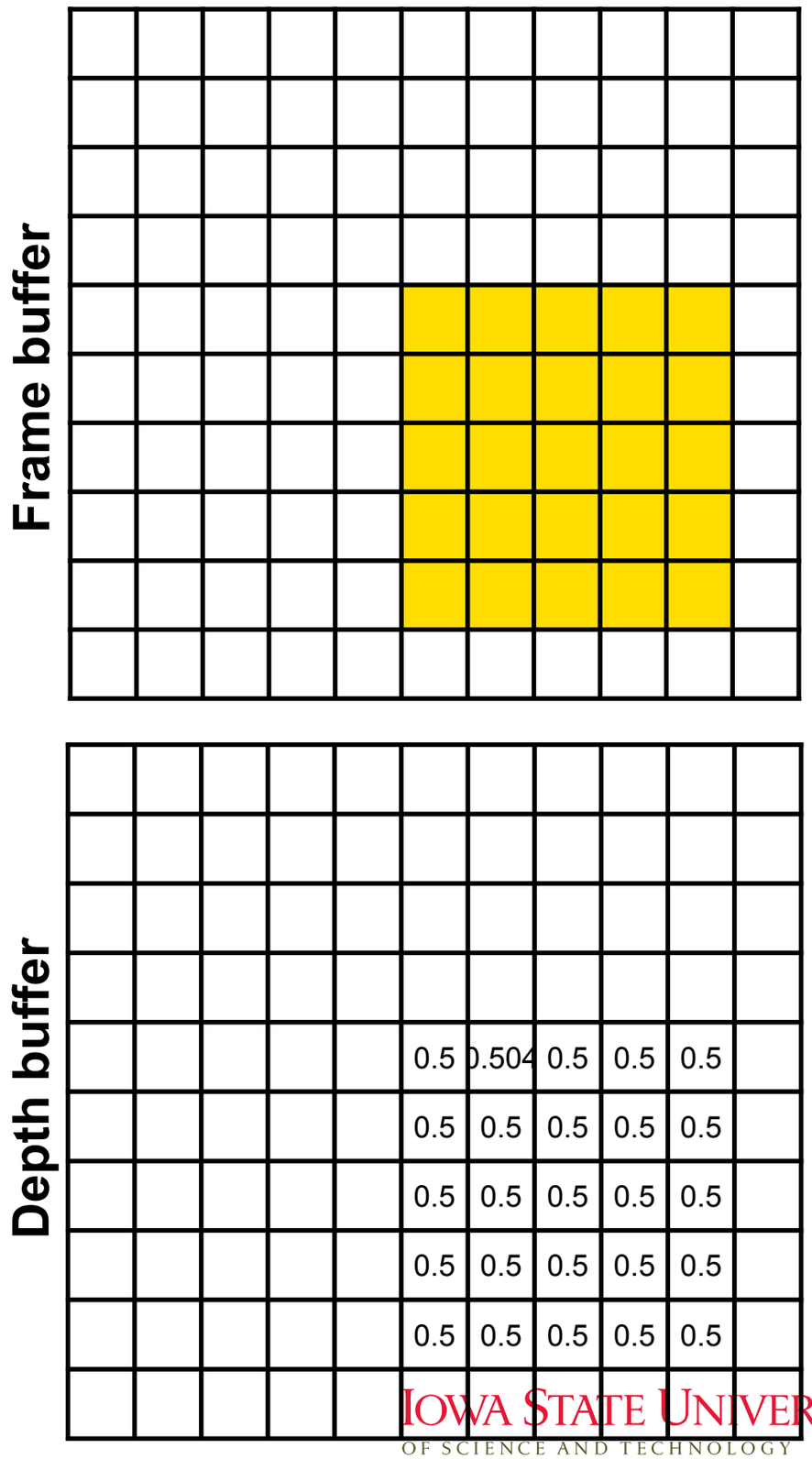


Why do we not see something like this?

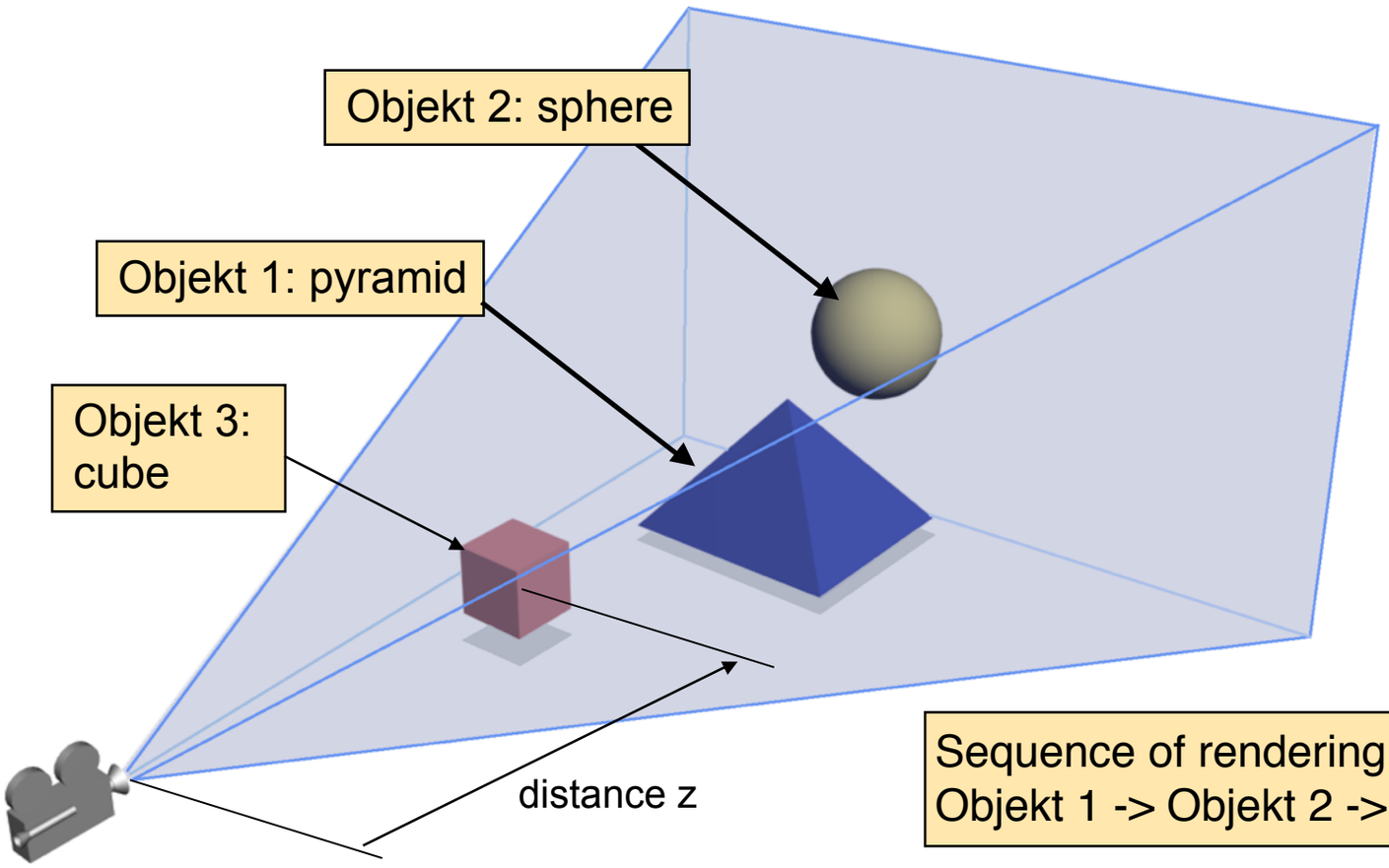
# Depth Buffer

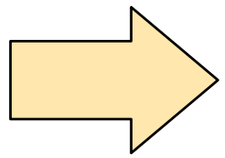


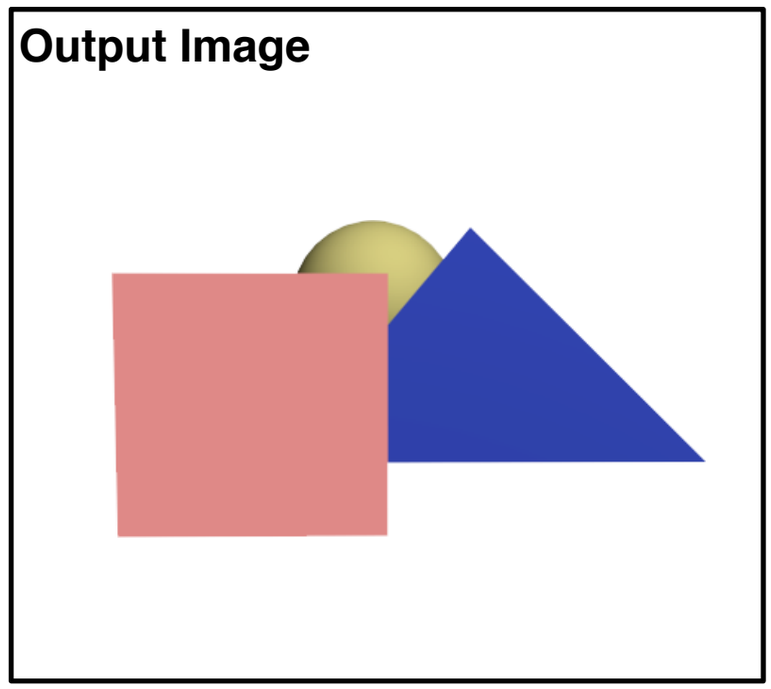

- Memory of the graphics hardware that keeps the distances values between objects (**per pixel**) on screen and the virtual camera.
- Same size in pixel than the frame buffer.
- Can only keep one value, usually you keep the distance to the closest object.
- Depth values are stored in a range between 0 (close) and 1 (far clipping plane)



# Depth Buffer Test



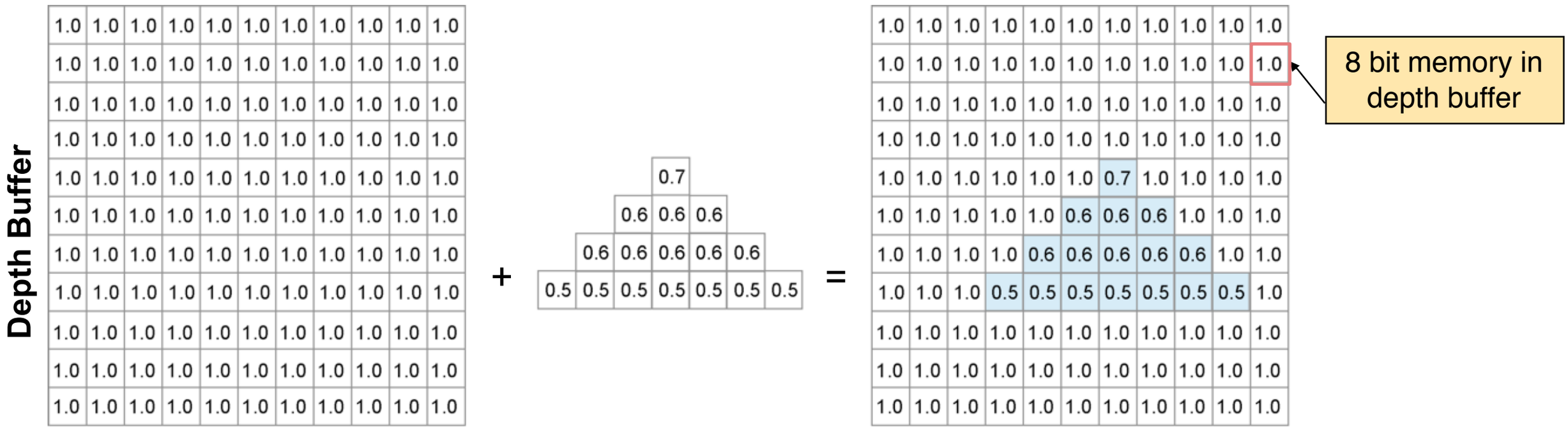



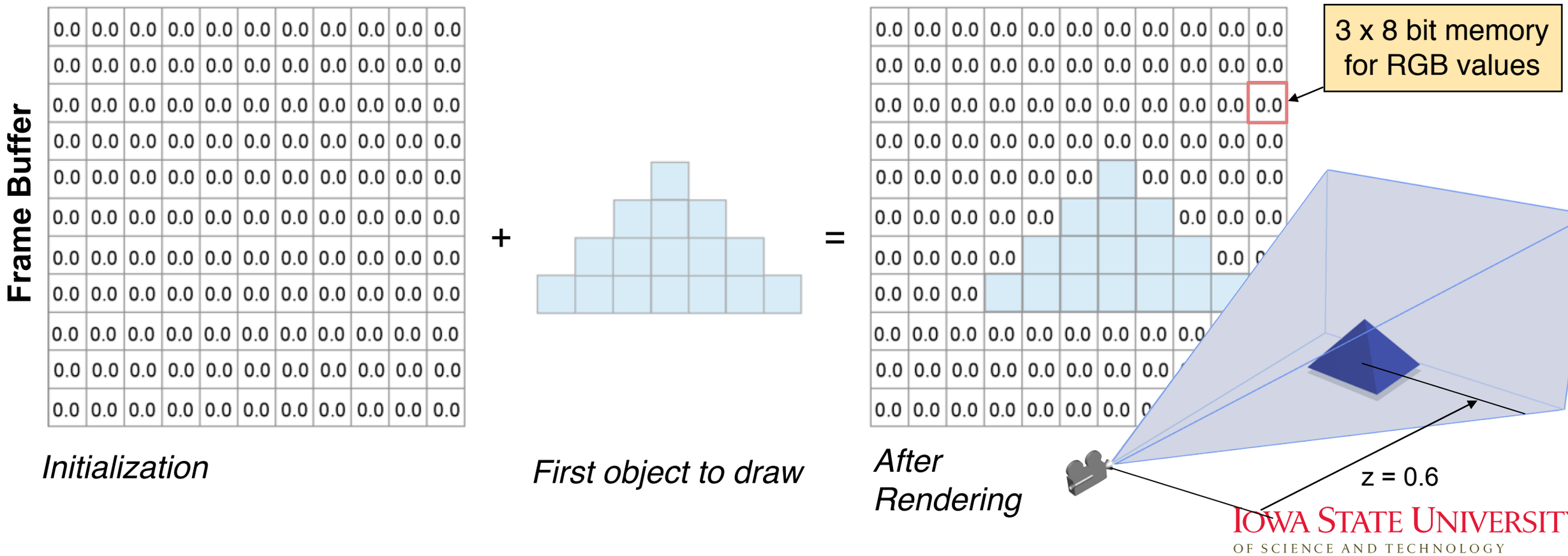



Sequence of rendering:
Objekt 1 -> Objekt 2 -> Objekt 3

- The Depth Buffer Test (of z-buffer test; developed by Cutmull) is used to test whether on object that should be rendered on screen is closer to the virtual camera than an object which has already been rendered into the frame buffer.
- Consider, the depth buffer keeps the depth information per pixel of the closet object to the virtual camera.
- A new object is only rendered into the frame buffer, if its pixel values are closer to the camera than the object which has already been rendered.

# Depth Buffer Example (1/3)




*Initialization*

*First object to draw*

*After Rendering*

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Depth Buffer Example (2/3)

VRAC | HCI

**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 | 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 | 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 | 1.0 | 1.0 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 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 | 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 | 1.0 | 1.0 |

+

*

| | | | | |
|---|---|---|---|---|
| | | 0.9 | | |
| | 0.9 | 0.9 | 0.9 | |
| 0.9 | 0.9 | 0.8 | 0.9 | 0.9 |
| | 0.9 | 0.9 | 0.9 | |
| | | 0.9 | | |

=

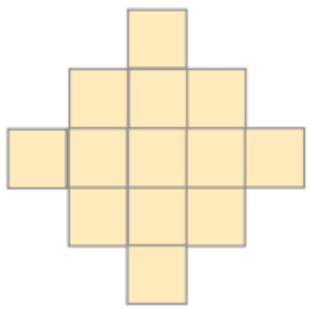
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.9 | 0.9 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 0.9 | 0.9 | 0.8 | 0.8 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.9 | 0.9 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 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 | 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 | 1.0 | 1.0 |

Parts of the second object remain hidden

**Frame Buffer**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | | | | | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | | | | | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

+



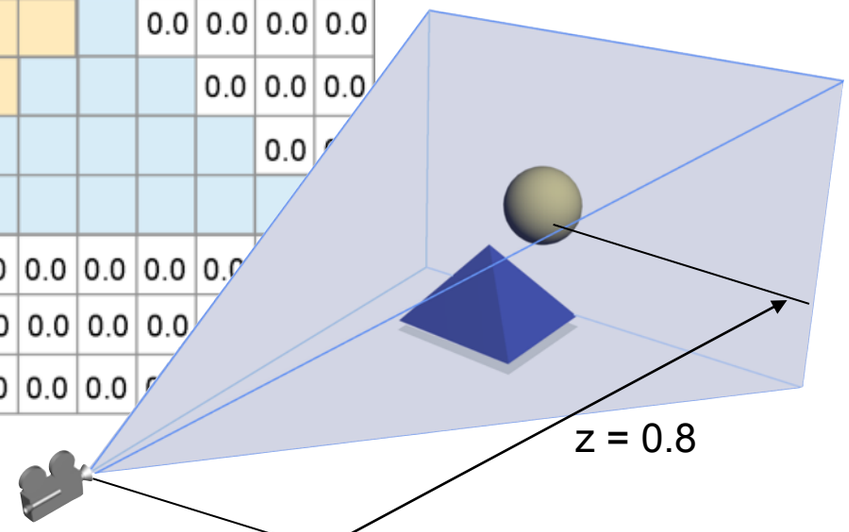
=

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | | | | | | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | | | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | | | | | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |

*After rendering of the first object*

*Second object*

*After Rendering*




* Note, the depth buffer stores no color values. The colors on this slide should only help to understand this concept

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Depth Buffer Example (3/3)


**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 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.9 | 0.9 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 0.9 | 0.9 | 0.8 | 0.9 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.9 | 0.9 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 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 | 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 | 1.0 | 1.0 |

+

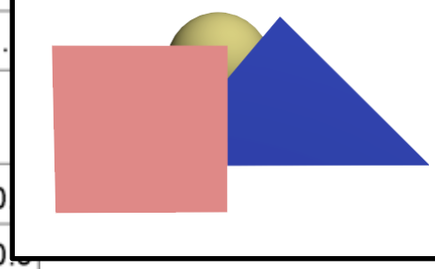
| | | | | |
|---|---|---|---|---|
| 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |

*

=

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 0.9 | 0.9 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.6 | 0.6 | 1.0 | 1.0 | 1.0 |
| 1.0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 |
| 1.0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.5 | 0.5 | 0.5 | 0.5 | 1. |
| 1.0 | 0.0 | 0.3 | 0.3 | 0.3 | 0.3 | 1.0 | 1.0 | 1.0 | 1.0 | 1. |
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1. |
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1. |

**Output Image**



**Frame Buffer**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | | | | | | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | | | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | | | | | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | | | | | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

+

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

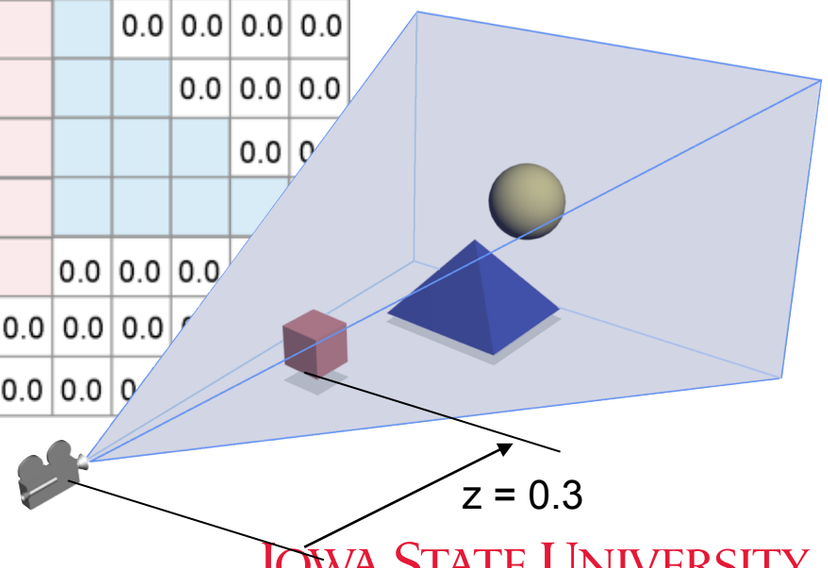
=

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| 0.0 | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | | | | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | | | | | | | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | | | | | | | | 0.0 | 0.0 | 0.0 |
| 0.0 | | | | | | | | | 0.0 | 0 |
| 0.0 | | | | | | | | | | |
| 0.0 | | | | | | 0.0 | 0.0 | 0.0 | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | | | |

*After rendering of the second object*

*Third object*

*After Rendering*




IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

* Note, the depth buffer stores no color values. The colors on this slide should only help to understand this concept

# Depth Buffer in OpenGL


Three things need to be done:

1. **During initialization: init the depth buffer memory.**
   The depth buffer is a memory. The program needs to know that it should use this memory.

2. **Enable the capability.**
   The depth buffer test is a capability of the fixed-function rendering pipeline. It must be enabled or disabled as required.

3. **Clear the buffer before you draw a new frame.**
   The buffer must be cleared before a new frame is rendered. Therefore, it has to be filled with the maximum value 1.0

# Code Example

Code from your main.cpp files

```
//// Main render loop

// Set up our green background color
static const GLfloat clear_color[] = { 0.6f, 0.7f, 1.0f, 1.0f };
static const GLfloat clear_depth[] = { 1.0f, 1.0f, 1.0f, 1.0f };

// This sets the camera to a new location
// the first parameter is the eye position, the second the center location, and the third the up vector.
SetViewAsLookAt(glm::vec3(1.0f, 1.0f, 2.5f), glm::vec3(1.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

```
// Enable depth test
// ignore this line, it allows us to keep the distance value after we proejct each object to a 2d canvas.
glEnable(GL_DEPTH_TEST);

// This is our render loop. As long as our window remains open (ESC is not pressed), we'll continue to render
    things.
while(!glfwWindowShouldClose(window))
{

    // Clear the entire buffer with our green color (sets the background to be green).
    glClearBufferfv(GL_COLOR , 0, clear_color);
    glClearBufferfv(GL_DEPTH , 0, clear_depth);
```

# Enable / Disable State Machine Functions

```
glEnable(GLenum cap);

glDisable(GLenum cap);
```

Enable or disable a capability of the graphics hardware.

**Parameter:**

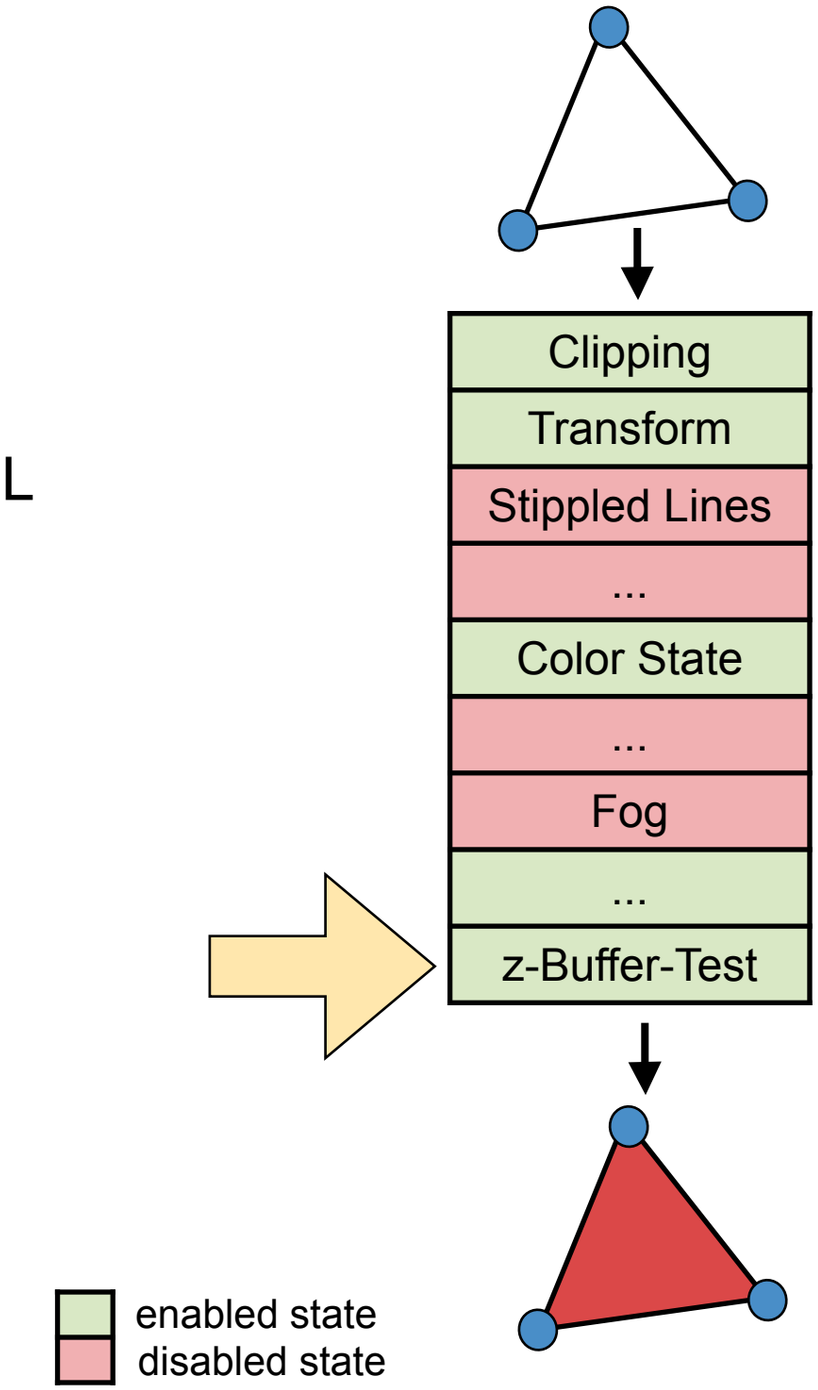
- cap: specifies a symbolic constant that indicates the GL capability

Examples:

GL_BLEND: If enabled, blend the computed fragment color values with the values in the color buffers.

**GL_DEPTH_TEST**: If enabled, do depth comparisons and update the depth buffer.

GL_STENCIL_TEST: If enabled, do stencil testing and update the stencil buffer.



# Thank you!

# Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY