ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Blending

October 6th, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

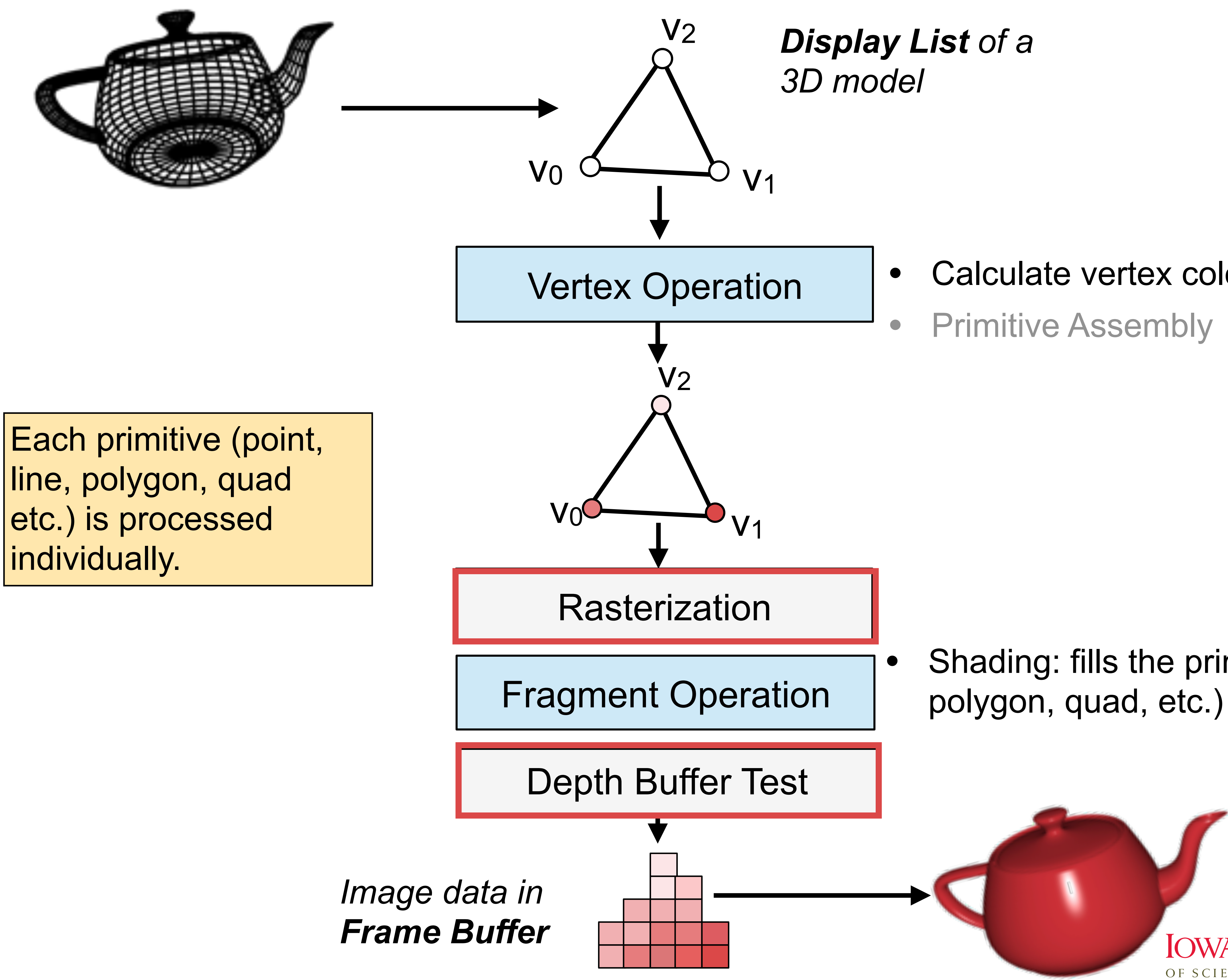
Add your name and the date on each document that you create and submit!

Documents without a name will not be graded anymore!

Content

- Blending in OpenGL
- Blending Function
- Blending Equation
- Rendering sequence

Rendering: from model to pixel

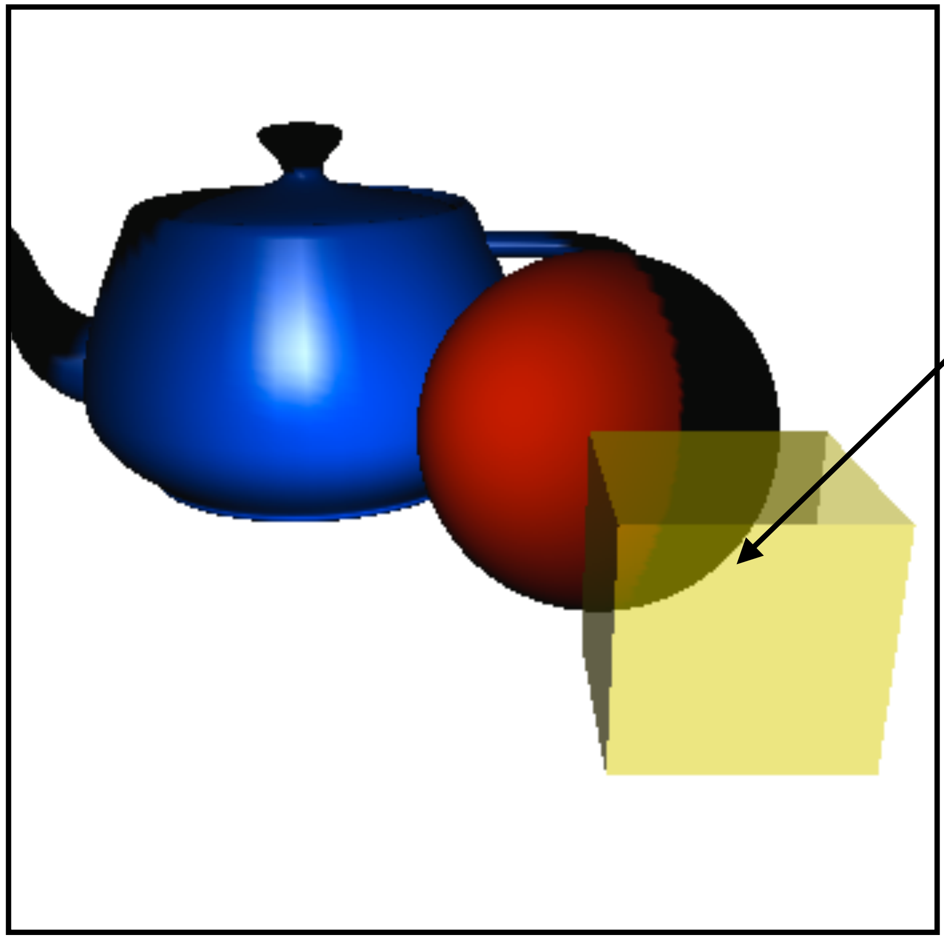


IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Blending

- Blending in OpenGL is used to blend the color of two or more objects.
- The function is applied before after the depth buffer test.
- Instead of removing the pixels that are already inside the buffer, the colors are mixed.

Why do we want to do something like this?



Output on display

Object transparency: the red sphere is visible through the yellow cube.

- In computer graphics, we simulate transparency by blending the color of objects.
- Blending is a capability of fixed-function rendering pipeline. It must be
  - enabled and a
  - blend value (alpha value) must be set.

Enable / Disable State Machine Functions

```
glEnable(GLenum cap);

glDisable(GLenum cap);
```

Enable or disable a capability of the graphics hardware.

Parameter:

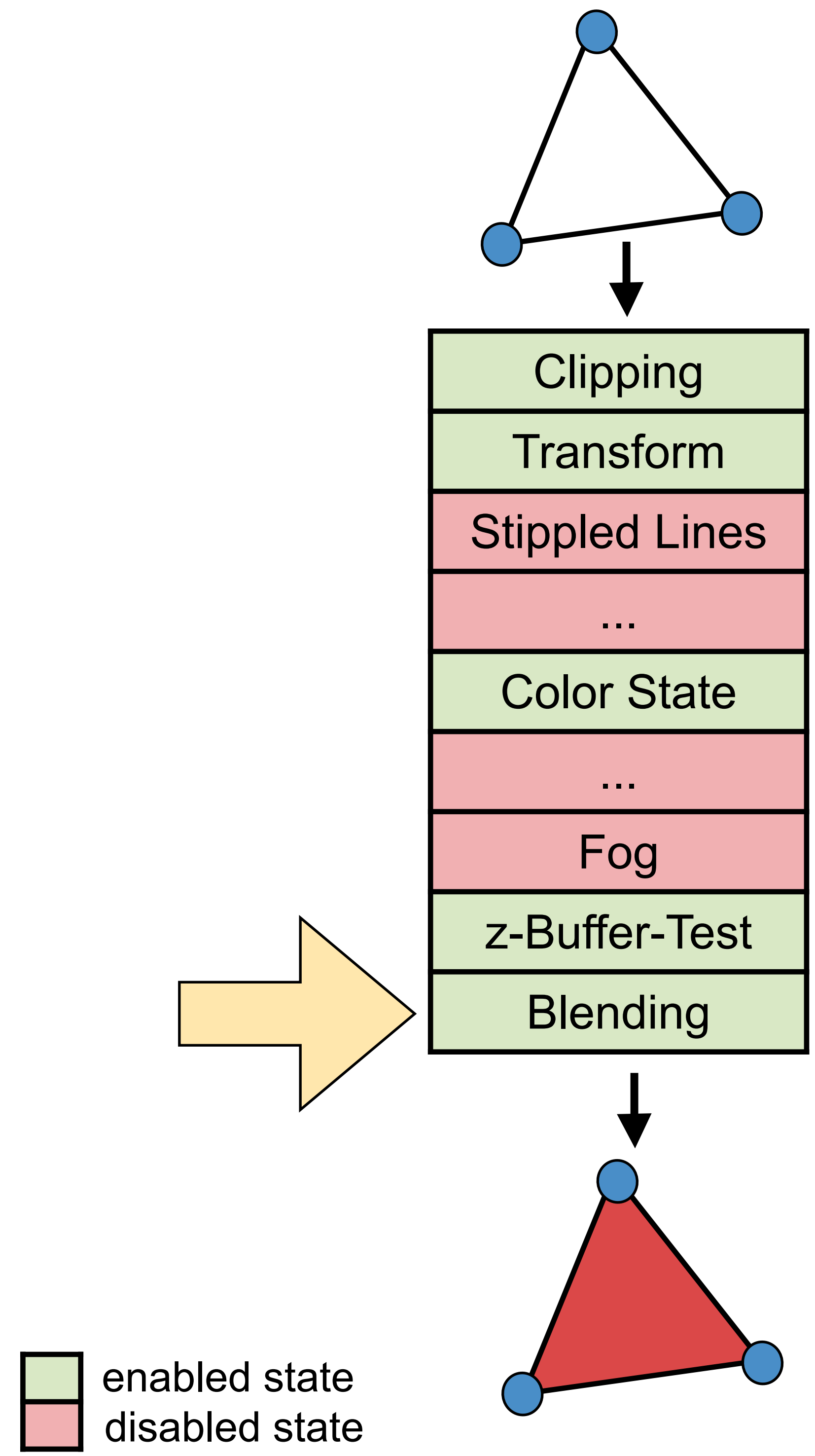
- cap: specifies a symbolic constant that indicates the GL capability

  Examples:

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

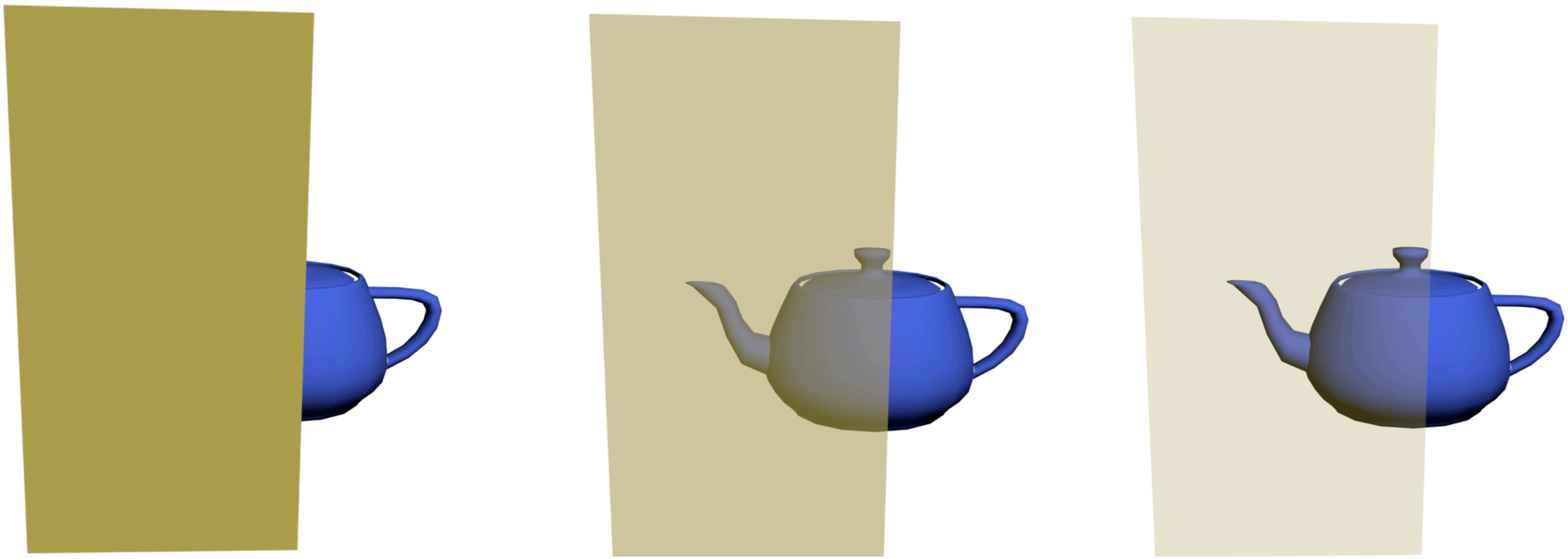
  GL_DEPTH_TEST: If enabled, do depth comparisons and update the depth buffer.

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



Alpha Value

The alpha value is a key value to determine the “amount of blending” between the object that is already inside the buffer and the object that should be rendered into the frame buffer



Alpha = 1.0 *Alpha = 0.6* *Alpha = 0.3*

```
// create a material for this cube
GLfloat yellow[] = {1.0, 1.0, 0.0, 0.5};
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, yellow);
```

Blend Function

```
void glBlendFunc( GLenum sfactor, GLenum dfactor);
```

Pixels can be drawn using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the frame buffer (the destination values). Blending is initially disabled.

Parameters:

- sfactor: Specifies how the red, green, blue, and alpha source blending factors are computed. The following symbolic constants are accepted: GL_ZERO, GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA. GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, and GL_ONE_MINUS_CONSTANT_ALPHA. The initial value is **GL_ONE**.
- dfactor: Specifies how the red, green, blue, and alpha destination blending factors are computed. The following symbolic constants are accepted: GL_ZERO, GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA. GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, and GL_ONE_MINUS_CONSTANT_ALPHA. The initial value is **GL_ZERO**.

Blend Function

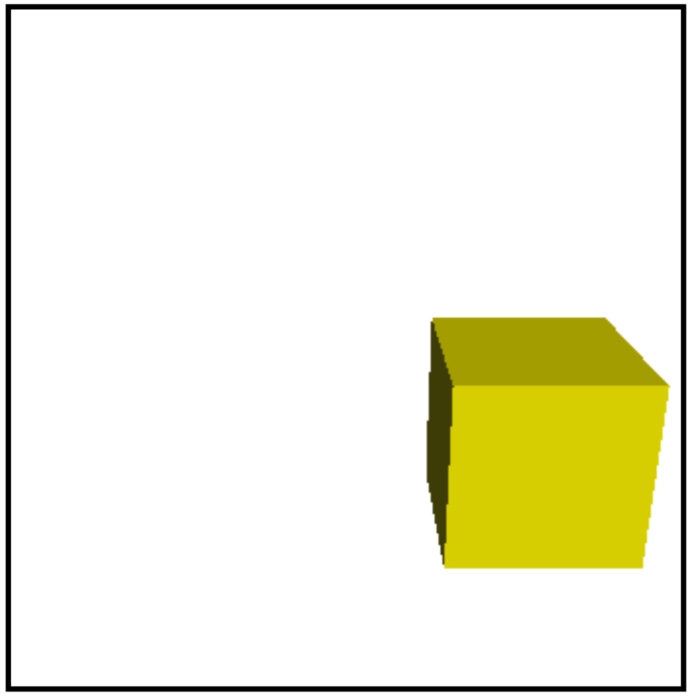
```
void glBlendFunc( GLenum sfactor, GLenum dfactor);
```

$$\begin{bmatrix} (sR \cdot [sfactor]) + (dR \cdot [dfactor]) \\ (sG \cdot [sfactor]) + (dG \cdot [dfactor]) \\ (sB \cdot [sfactor]) + (dB \cdot [dfactor]) \\ (sA \cdot [sfactor]) + (dA \cdot [dfactor]) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

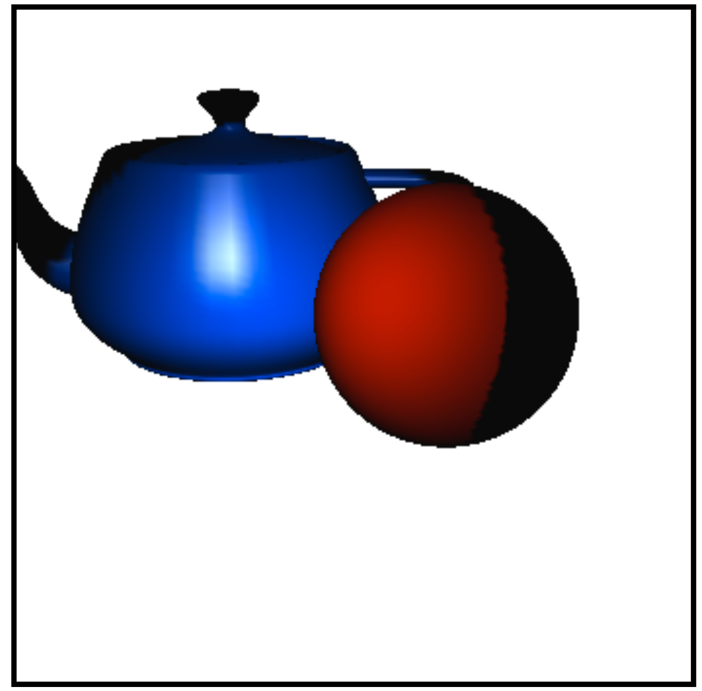
The blend function is the equation that is used to blend the color

- sR, sG, SB, sA: source; pixel color of the object that need to be rendered to frame buffer
- dR, dG, dB, dA: destination; pixel color that is already inside the frame buffer
- rR, rG, rB, rA: result; the new color which is rendered into the frame buffer

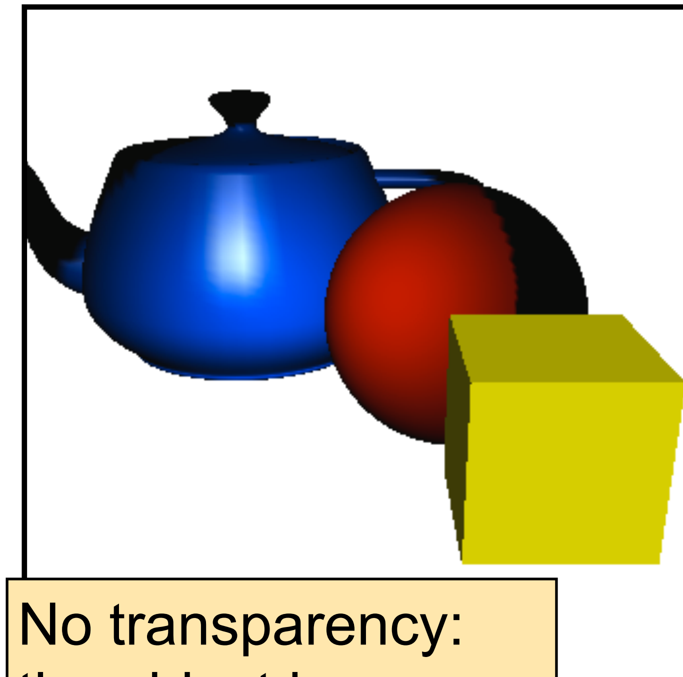
Blend Function Example



+



=



No transparency: the object is rendered with its original color

Need to be added

Already inside the frame buffer

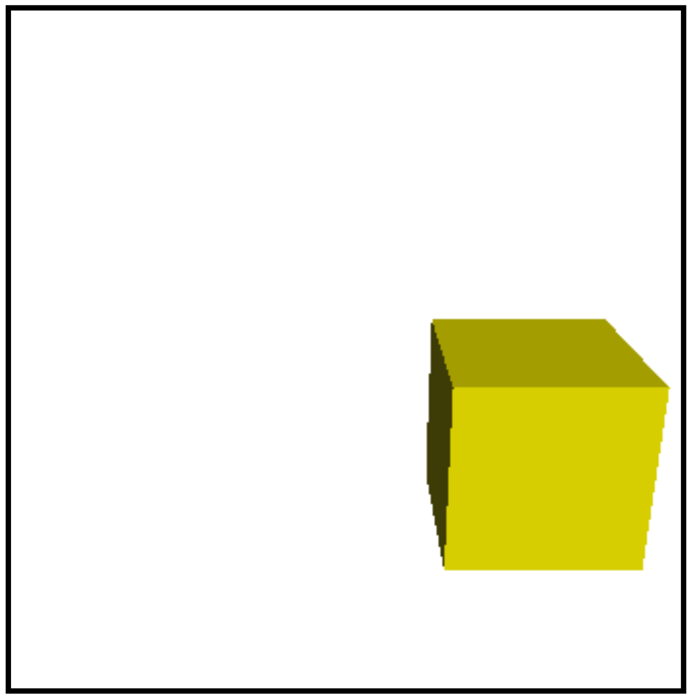
```
glBlendFunc( GL_ONE, GL_ZERO);
```

$$\begin{bmatrix} (sR \cdot 1) + (dR \cdot 0) \\ (sG \cdot 1) + (dG \cdot 0) \\ (sB \cdot 1) + (dB \cdot 0) \\ (sA \cdot 1) + (dA \cdot 0) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

s: source *d: destination* *r: result*

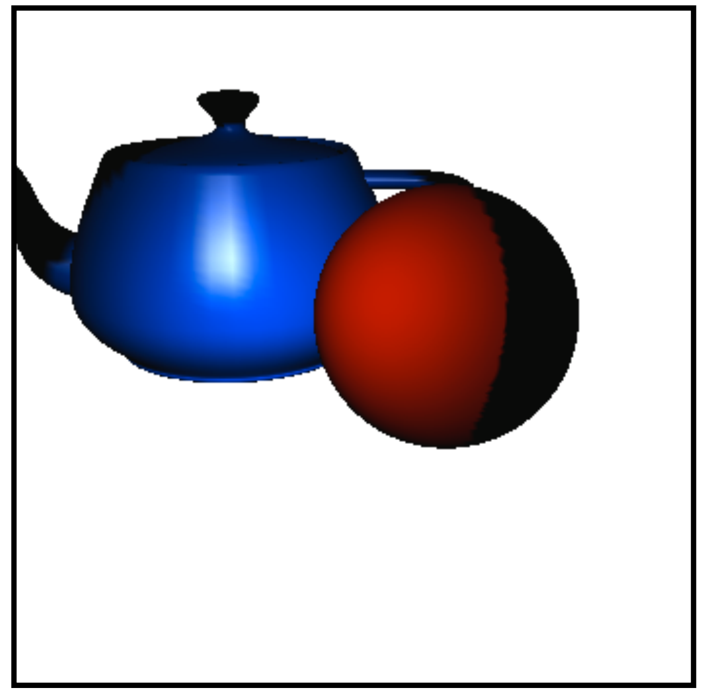
```
GL_ZERO: 0
GL_ONE: 1
```

Blend Function Example



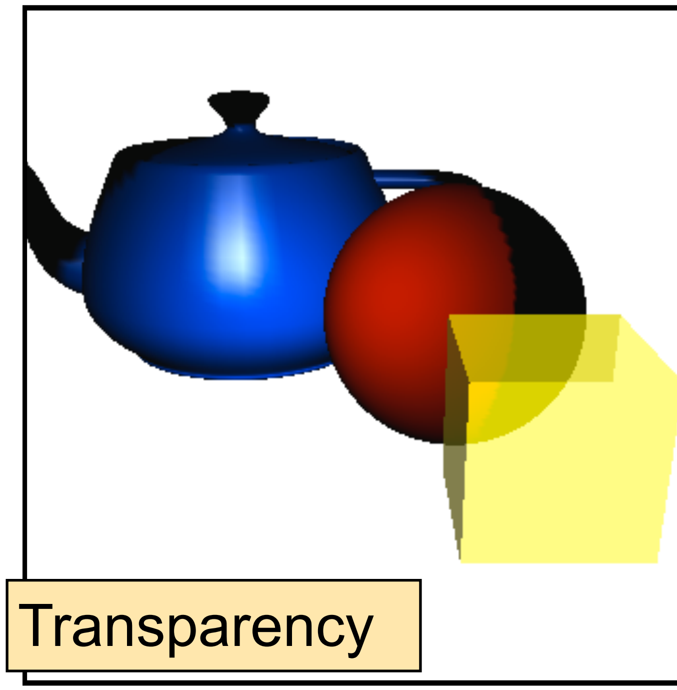
Need to be added

+



Already inside the frame buffer

=



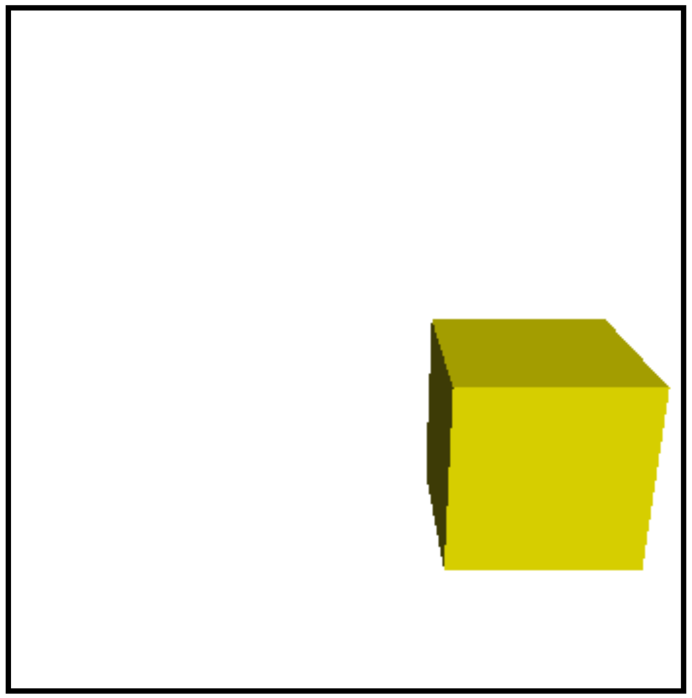

```
glBlendFunc( GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
```

$$\begin{bmatrix} (sR \cdot 1) + (dR \cdot (1 - sA)) \\ (sG \cdot 1) + (dG \cdot (1 - sA)) \\ (sB \cdot 1) + (dB \cdot (1 - sA)) \\ (sA \cdot 1) + (dA \cdot (1 - sA)) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

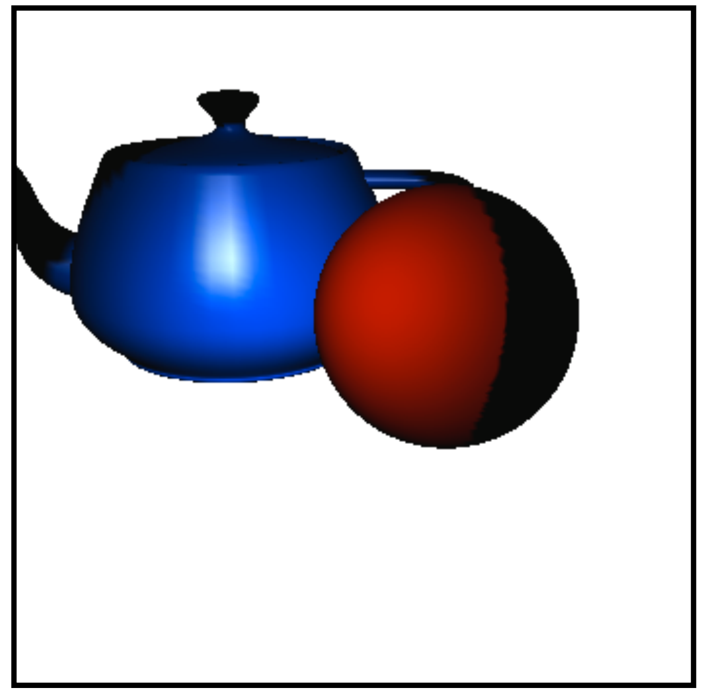
s: source   d: destination   r: result

```
GL_ONE: 1
GL_ONE_MINUS_SRC_ALPHA: 1 – sA
```

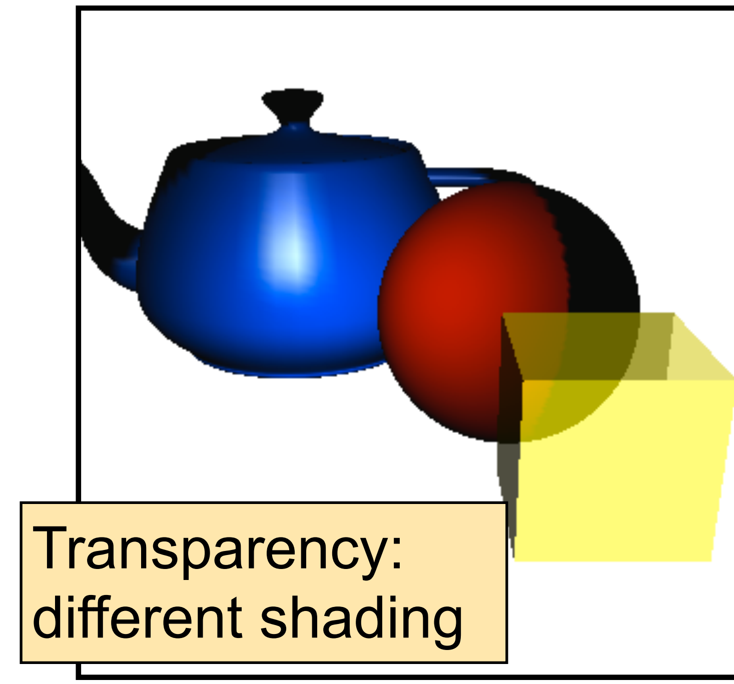
Blend Function Example



+



=



Need to be added

Already inside the frame buffer

```
glBlendFunc( GL_SRC_COLOR, GL_ONE_MINUS_SRC_ALPHA );
```

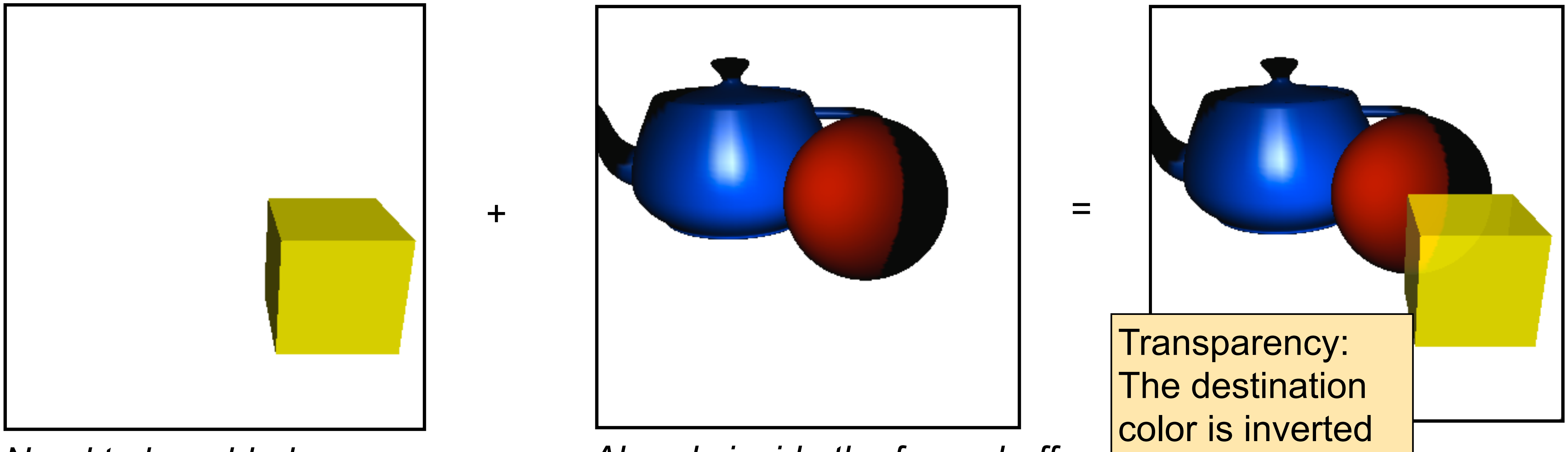
$$\begin{bmatrix} (sR \cdot sR) + (dR \cdot (1 - sA)) \\ (sG \cdot sG) + (dG \cdot (1 - sA)) \\ (sB \cdot sB) + (dB \cdot (1 - sA)) \\ (sA \cdot sA) + (dA \cdot (1 - sA)) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

s: source *d: destination* *r: result*

`GL_SRC_COLOR: sR, sG, sB, sA`

`GL_ONE_MINUS_SRC_ALPHA: 1 – sA`

Blend Function Example



Need to be added

Already inside the frame buffer

```
glBlendFunc( GL_ONE, GL_ONE_MINUS_DST_COLOR);
```

$$\begin{bmatrix} (sR \cdot \boxed{1}) + (dR \cdot \boxed{(1 - dR)}) \\ (sG \cdot 1) + (dG \cdot (1 - dG)) \\ (sB \cdot 1) + (dB \cdot (1 - dB)) \\ (sA \cdot 1) + (dA \cdot (1 - dA)) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

`GL_ONE: 1`

`GL_ONE_MINUS_DST_COLOR: 1 – d[RGB]`

s: source *d: destination* *r: result*

glBlendEquation

```
void glBlendEquation(GLenum mode)
```

The blend equations determine how a new pixel (the "source" color) is combined with a pixel already in the framebuffer (the "destination" color).

Parameters:

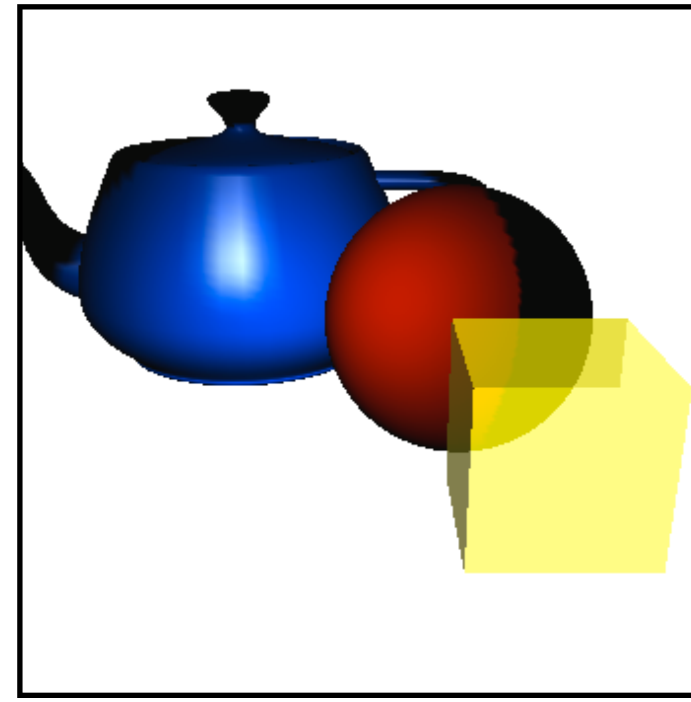
- mode: specifies how source and destination colors are combined.
  It must be GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_SUBTRACT, GL_MIN, GL_MAX.

Blend Equation Example

glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);

glBlendEquation(GL_FUNC_ADD);

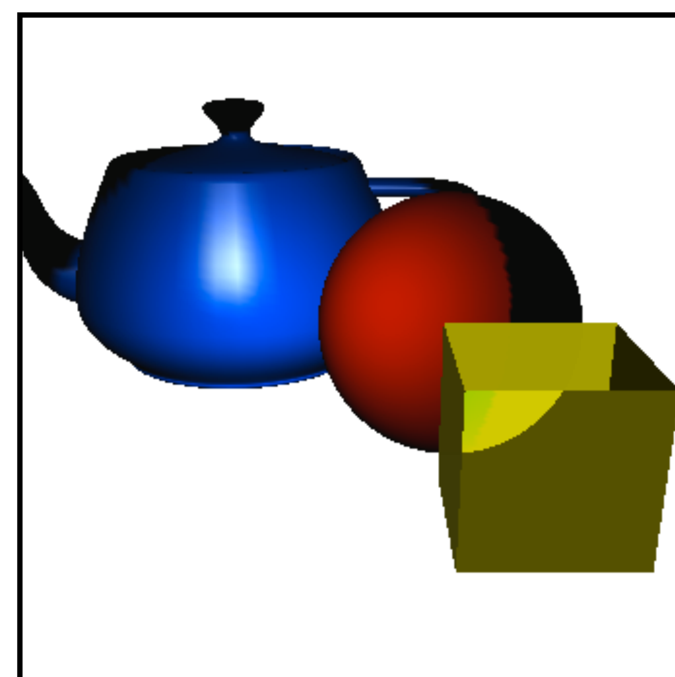
$$\begin{bmatrix} (sR \cdot 1) + (dR \cdot (1 - sA)) \\ (sG \cdot 1) + (dG \cdot (1 - sA)) \\ (sB \cdot 1) + (dB \cdot (1 - sA)) \\ (sA \cdot 1) + (dA \cdot (1 - sA)) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$



glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);

glBlendEquation(GL_FUNC_SUBTRACT);

$$\begin{bmatrix} (sR \cdot 1) - (dR \cdot (1 - sA)) \\ (sG \cdot 1) - (dG \cdot (1 - sA)) \\ (sB \cdot 1) - (dB \cdot (1 - sA)) \\ (sA \cdot 1) - (dA \cdot (1 - sA)) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

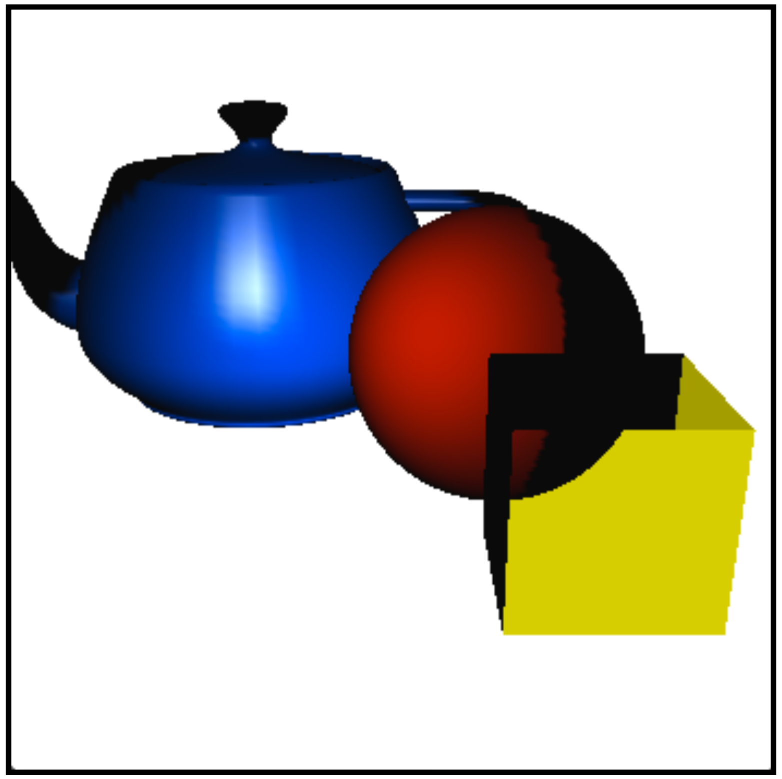


Blend Equation Example

glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);

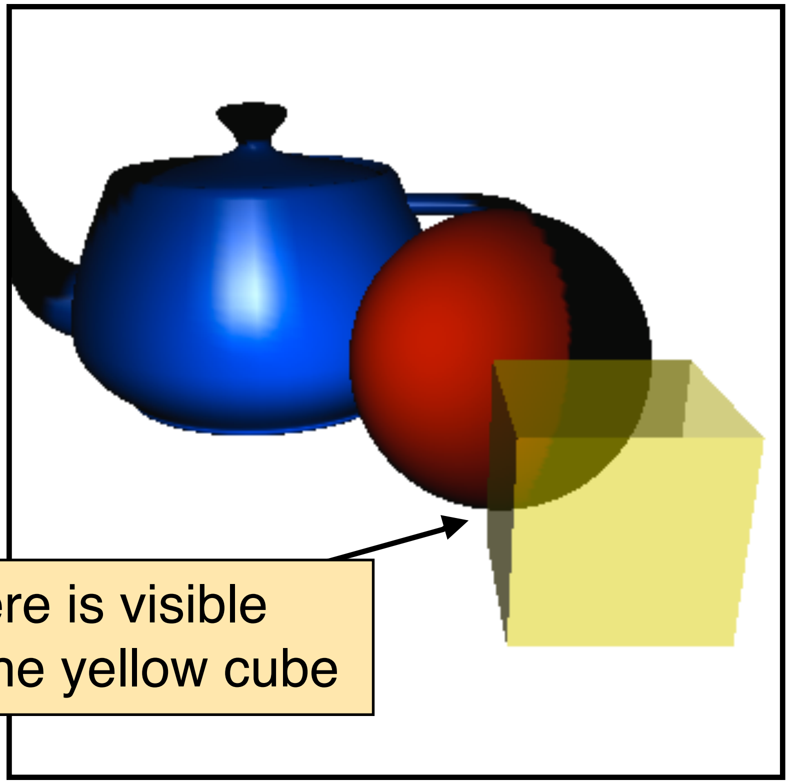
glBlendEquation(GL_MIN);

$$\begin{bmatrix} \boxed{min}(sR, dR) \\ min(sG, dG) \\ min(sB, dB) \\ min(sA, dA) \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$



- Not all combinations are useful. You need to decide on your own which results you want to achieve.
- The “common” transparency of e.g. a window is *glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)* *glBlendEquation(GL_ADD);*

Rendering Sequence

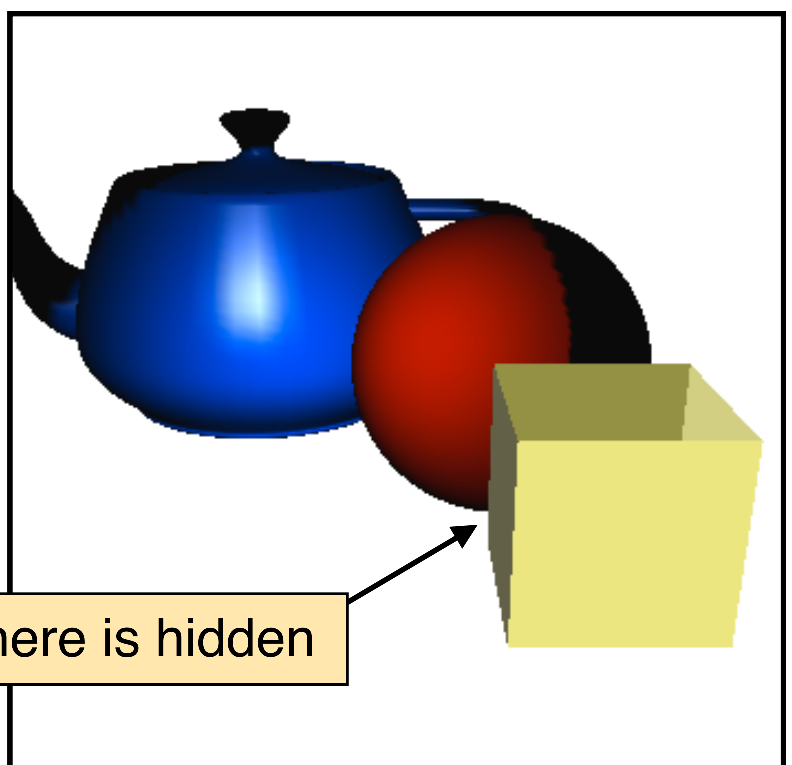



```
void draw_scene(void)
{
    glEnable(GL_DEPTH_TEST);

    // draw a solid sphere
    draw_solid_sphere();

    // draw a teapot
    draw_solid_teapot();

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




```
void draw_scene(void)
{
    glEnable(GL_DEPTH_TEST);

    // draw solid cube
    draw_solid_cube();

    // draw a solid sphere
    draw_solid_sphere();

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

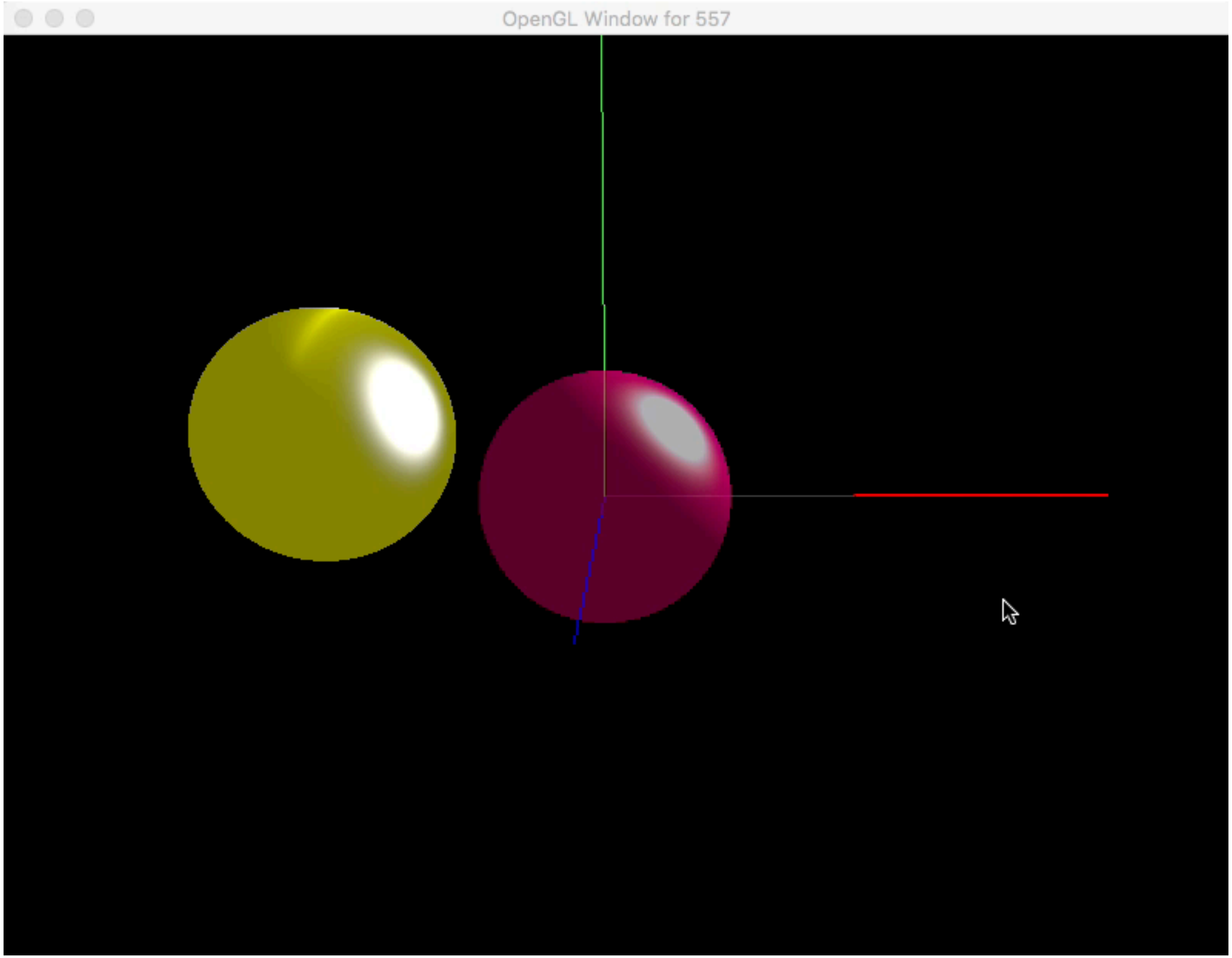
The depth test considers the rendering order of objects. The object that should appear through a transparent object must already be in the frame buffer before the transparent object is drawn. Otherwise, the object behind the transparent object will disappear.

Reason: the red sphere will not pass the z-buffer test. All pixels, which are behind the yellow cube are removed in this test; the pixels of the yellow cube have smaller z-values than the pixels of the red sphere.

Render sequence:

1. Render all opaque objects
2. Render all transparent objects

Video



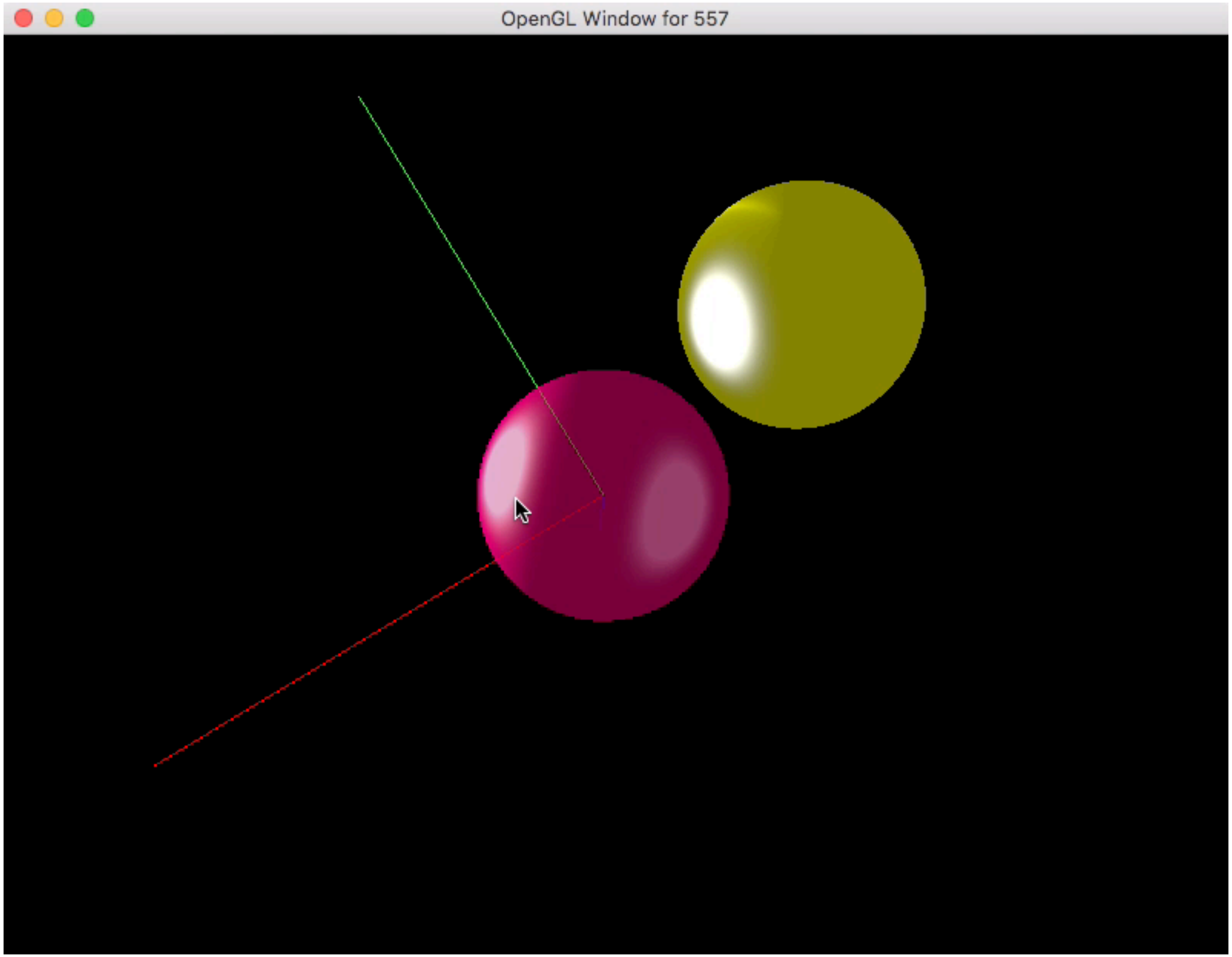
Two objects in the correct rendering order

```
while ...
{

   // draw a yellow sphere
   draw_yellow_sphere();

   // draw the red sphere
   draw_red_sphere();
}
```

Video



Two objects in the wrong rendering order

```
while ...
{

  // draw the red sphere
  draw_red_sphere();

  // draw a yellow sphere
  draw_yellow_sphere();

}
```

Reference Table

Parameter	(f_R, f_G, f_B, f_A)
GL_ZERO	$(0,0,0,0)$
GL_ONE	$(1,1,1,1)$
GL_SRC_COLOR	$\left(\frac{R_{s0}}{k_R}, \frac{G_{s0}}{k_G}, \frac{B_{s0}}{k_B}, \frac{A_{s0}}{k_A}\right)$
GL_ONE_MINUS_SRC_COLOR	$(1,1,1,1) - \left(\frac{R_{s0}}{k_R}, \frac{G_{s0}}{k_G}, \frac{B_{s0}}{k_B}, \frac{A_{s0}}{k_A}\right)$
GL_DST_COLOR	$\left(\frac{R_d}{k_R}, \frac{G_d}{k_G}, \frac{B_d}{k_B}, \frac{A_d}{k_A}\right)$
GL_ONE_MINUS_DST_COLOR	$(1,1,1,1) - \left(\frac{R_d}{k_R}, \frac{G_d}{k_G}, \frac{B_d}{k_B}, \frac{A_d}{k_A}\right)$
GL_SRC_ALPHA	$\left(\frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}\right)$
GL_ONE_MINUS_SRC_ALPHA	$(1,1,1,1) - \left(\frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}, \frac{A_{s0}}{k_A}\right)$
GL_DST_ALPHA	$\left(\frac{A_d}{k_A}, \frac{A_d}{k_A}, \frac{A_d}{k_A}, \frac{A_d}{k_A}\right)$
GL_ONE_MINUS_DST_ALPHA	$(1,1,1,1) - \left(\frac{A_d}{k_A}, \frac{A_d}{k_A}, \frac{A_d}{k_A}, \frac{A_d}{k_A}\right)$
GL_CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
GL_ONE_MINUS_CONSTANT_COLOR	$(1,1,1,1) - (R_c, G_c, B_c, A_c)$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)
GL_ONE_MINUS_CONSTANT_ALPHA	$(1,1,1,1) - (A_c, A_c, A_c, A_c)$
GL_SRC_ALPHA_SATURATE	$(i,i,i,1)$

GL_SRC1_COLOR	$\left(\frac{R_{s1}}{k_R}, \frac{G_{s1}}{k_G}, \frac{B_{s1}}{k_B}, \frac{A_{s1}}{k_A}\right)$
GL_ONE_MINUS_SRC1_COLOR	$(1,1,1,1) - \left(\frac{R_{s1}}{k_R}, \frac{G_{s1}}{k_G}, \frac{B_{s1}}{k_B}, \frac{A_{s1}}{k_A}\right)$
GL_SRC1_ALPHA	$\left(\frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}\right)$
GL_ONE_MINUS_SRC1_ALPHA	$(1,1,1,1) - \left(\frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}, \frac{A_{s1}}{k_A}\right)$

Mode	RGB Components	Alpha Component
GL_FUNC_ADD	$Rr = R_s s_R + R_d d_R$ $Gr = G_s s_G + G_d d_G$ $Br = B_s s_B + B_d d_B$	$Ar = A_s s_A + A_d d_A$
GL_FUNC_SUBTRACT	$Rr = R_s s_R - R_d d_R$ $Gr = G_s s_G - G_d d_G$ $Br = B_s s_B - B_d d_B$	$Ar = A_s s_A - A_d d_A$
GL_FUNC_REVERSE_SUBTRACT	$Rr = R_d d_R - R_s s_R$ $Gr = G_d d_G - G_s s_G$ $Br = B_d d_B - B_s s_B$	$Ar = A_d d_A - A_s s_A$
GL_MIN	$Rr = min(R_s, R_d)$ $Gr = min(G_s, G_d)$ $Br = min(B_s, B_d)$	$Ar = min(A_s, A_d)$
GL_MAX	$Rr = max(R_s, R_d)$ $Gr = max(G_s, G_d)$ $Br = max(B_s, B_d)$	$Ar = max(A_s, A_d)$

Note, all values k are a so called blend value. Consider it as 1.0; we discuss it later in class

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