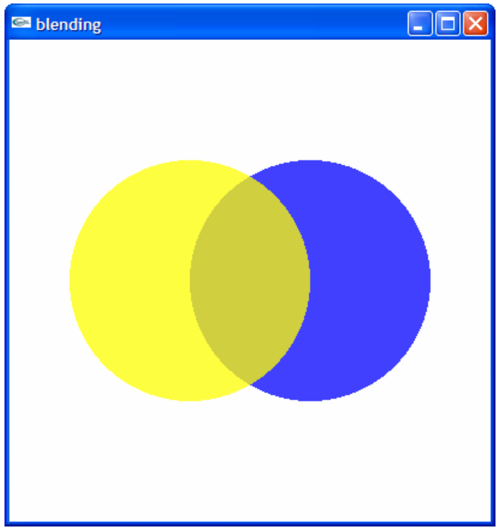
OpenGL® Lectures
Transparency Case Study

By
Tom Duff
Pixar Animation Studios
Emeryville, California
and
George Ledin Jr
Sonoma State University
Rohnert Park, California

How is transparency achieved in OpenGL?

- In OpenGL, we use blending of alpha value (opacity value) to create a translucent fragment that lets some of the previously stored color value “show through”.



Enable/Disable blending of color

- To enable transparency, we need to explicitly enable blending:
  - glEnable(GL_BLEND)

- To disable blending:
  - glDisable(GL_BLEND)

Alpha Value

- void **glClearColor**(GLclampf *red*, GLclampf *green*, GLclampf *blue*, GLclampf *alpha*)
  - Specify values for the color buffer

- void **glColor4f**(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*)
  - Specify colors

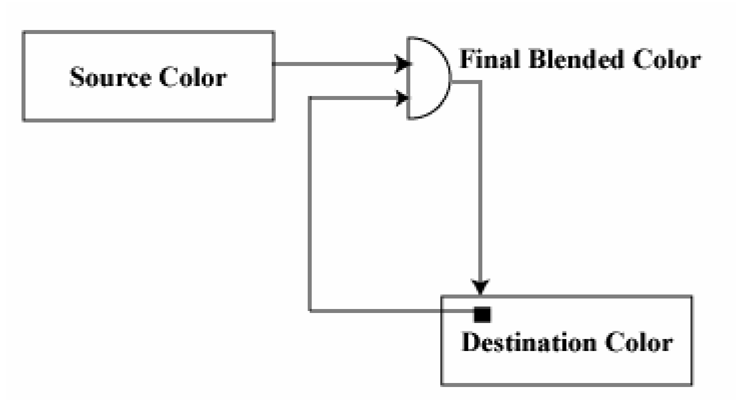
0 <= alpha <= 1

Transparent **opaque**

What happens when blending is enabled?

- When blending is enabled, the alpha value is used to combine the color value of the fragment being processed with that of the pixel already stored in the framebuffer.
  - Blending occurs after the scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the framebuffer.
- Without blending, each new fragment overwrites any existing color values in the frame buffer, as if the fragment were opaque.
- With blending, one can control how much of the existing color value should be combined with the new fragment's value.

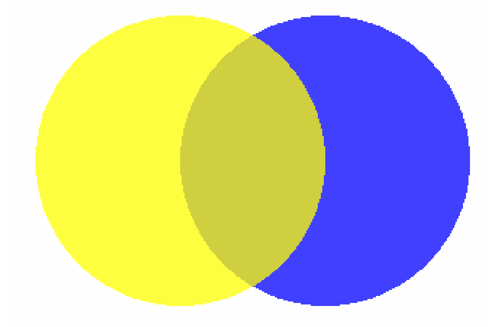
Blending Model



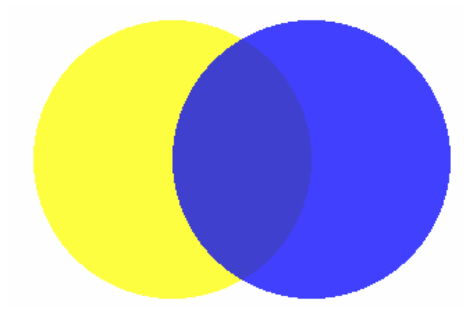
Case Study 1
a real world scenario

- Scenario:
  - There's a yellow circle and a blue circle.
    The yellow circle is 25% transparent. The blue circle is 75% transparent.
- Question:
  - When the yellow circle is in front of the blue circle, what do we see, Choice A or Choice B?

- Choice A



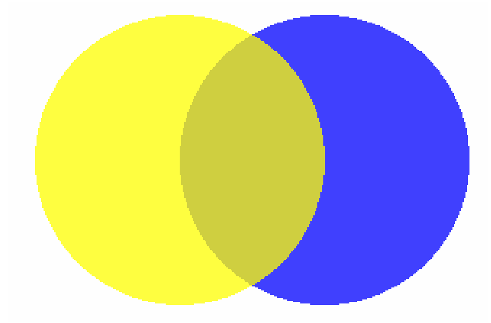
- Choice B



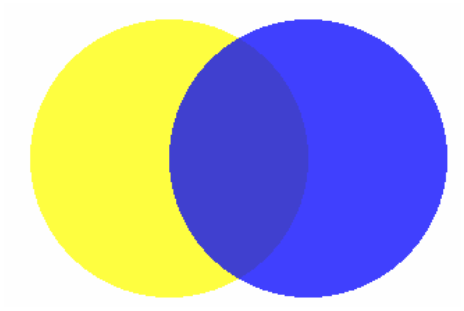
Case Study 1 – Answer is A
Reason: Because yellow circle is more opaque, therefore we see more of the yellow and less of the blue.

- Scenario:
  - There's a yellow circle and blue circle.
    - The yellow circle is 25% transparent. The blue circle is 75% transparent.
  - This means that the yellow circle is 75% opaque and blue circle is 25% opaque.
- If the yellow circle is in front of the blue circle, only 25% of the blue blends with 75% of the yellow. (Choice A)
- If the blue circle is in front of the yellow circle, only 25% of yellow blends with 75% of the blue. (Choice B)

- Choice A



- Choice B

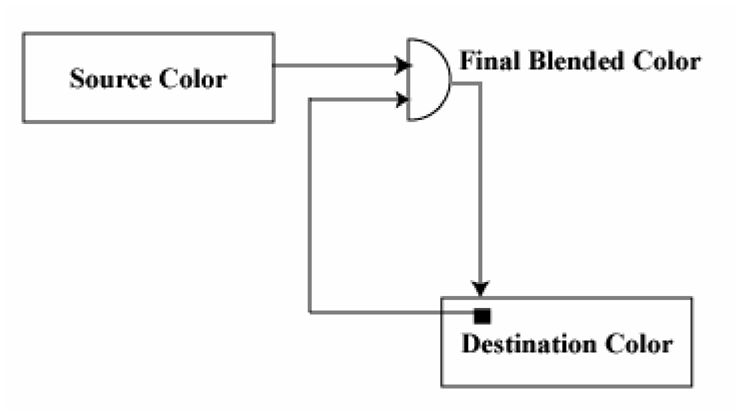


The source and destination factors

- During blending, source is the color value of the incoming fragment.

- Destination is the currently stored pixel value.

Blending Model



How does OpenGL blend the source and destination color values?

- Assume:
  - Source Color: (Rs, Gs, Bs, As)
  - Destination Color: (Rd, Gd, Bd, Ad)

- Step 1:
  - Specify the source and destination factors.
    - Let source destination blending factors be:
      - (Sr, Sg, Sb, Sa)

    - Let destination blending factors be:
      - (Dr, Dg, Db, Da)

- Step 2:
  - Combine the corresponding components of source and destination.

    - Final blended RGBA values are given by:
      - (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)

- Note: Each of these quadruplets is clamped to [0,1]

Blend Function

- void **glBlendFunc**(GLenum *sfactor*, GLenum *dfactor*)

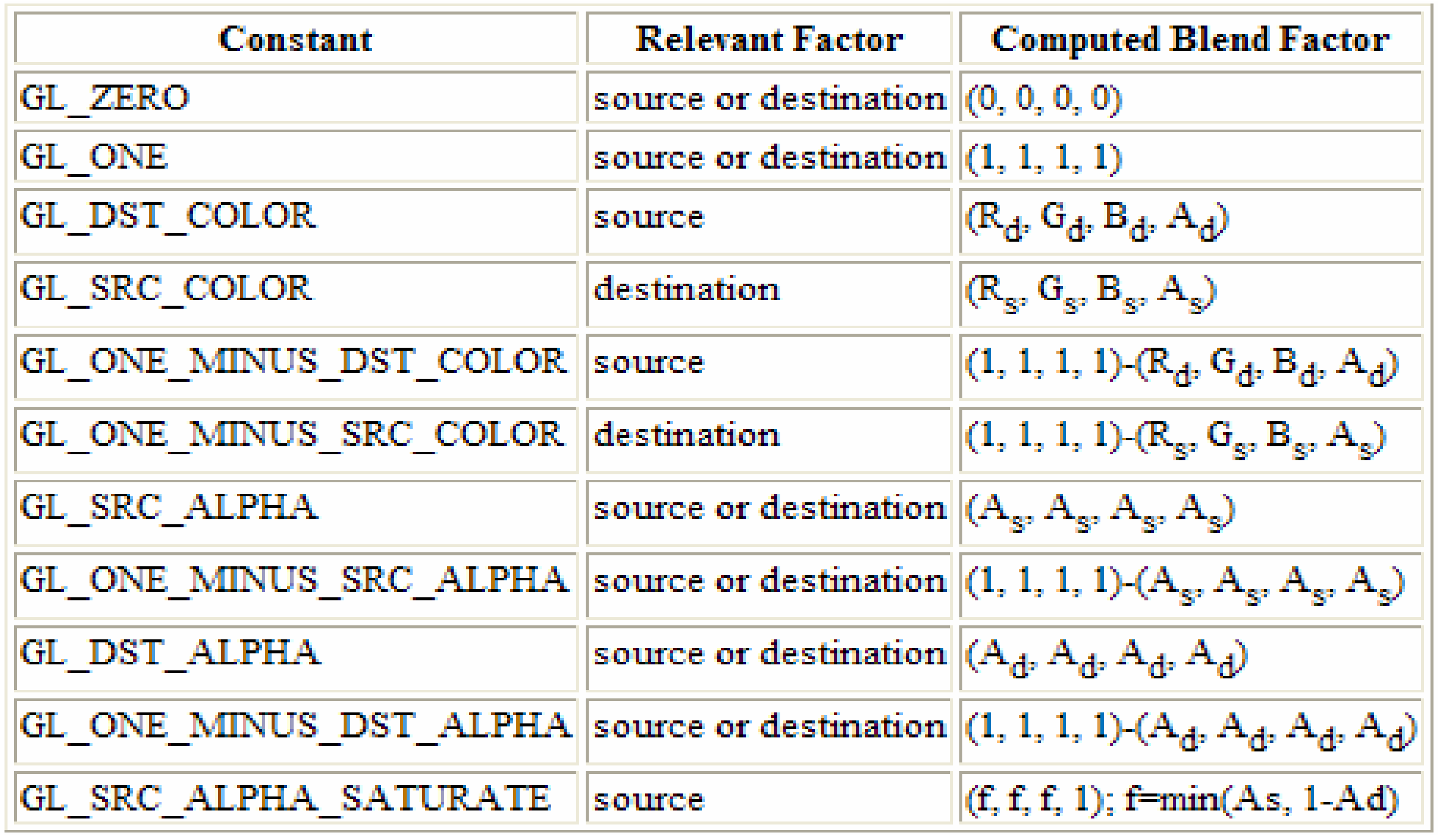
- *sfactor*
  - Specifies how the red, green, blue and alpha source blending factors are computed. Nine symbolic constants are accepted:
    - GL_ZERO
    - GL_ONE
    - **GL_DST_**
    - **COLOR**
    - **GL_ONE_MINUS_DST_COLOR**
    - GL_SRC_ALPHA
    - GL_ONE_MINUS_SRC_COLOR
    - GL_DST_ALPHA
    - GL_ONE_MINUS_DST_ALPHA
    - **GL_SRC_ALPHA_SATURATE**

- *dfactor*
  - Specifies how the red, green, blue and alpha destination blending factors are computed. Eight symbolic constants are accepted:
    - GL_ZERO
    - GL_ONE
    - **GL_SCR_COLOR**
    - **GL_ONE_MINUS_SRC_COLOR**
    - GL_SRC_ALPHA
    - GL_ONE_MINUS_SRC_COLOR
      ,
    - GL_DST_ALPHA
      GL_ONE_MINUS_DST_ALPHA.

- Blending factors lie in the range [0,1]. After the color values in the source and destination are combined, they're clamped to the range [0,1].

Source and Destination Blending Factors Table

sFactor=(Sr,Sg,Sb,Sa) or dFactor=(Dr,Dg,Db,Da)



Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(R_d, G_d, B_d, A_d)
GL_SRC_COLOR	destination	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_DST_COLOR	source	$(1, 1, 1, 1)-(R_d, G_d, B_d, A_d)$
GL_ONE_MINUS_SRC_COLOR	destination	$(1, 1, 1, 1)-(R_s, G_s, B_s, A_s)$
GL_SRC_ALPHA	source or destination	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	source or destination	$(1, 1, 1, 1)-(A_s, A_s, A_s, A_s)$
GL_DST_ALPHA	source or destination	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	source or destination	$(1, 1, 1, 1)-(A_d, A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

Case Study 2
Source and Destination Blending Factors

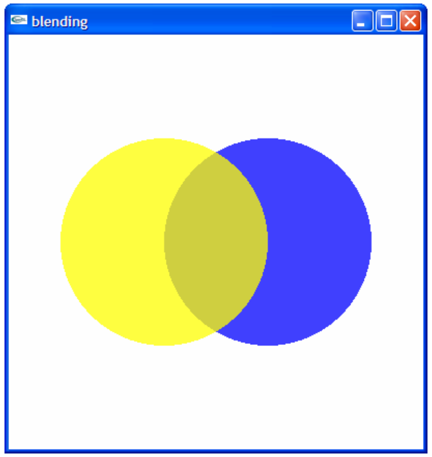
- Case Study Setup:
  - A yellow circle of radius 2, centered at (-1,0,0)

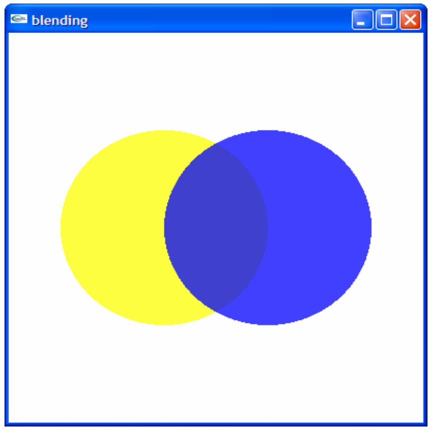
```
static void drawLeftCircle(void){
        /* draw yellow triangle on LHS of screen */
        glColor4f(1.0, 1.0, 0.0, 0.75); //yellow, alpha=0.75
        circle(-1,0,2); //x=-1,y=0,radius=2
        }
```

  - A blue circle of radius 2, centered at (1,0,0)

```
static void drawRightCircle(void){
        /* draw yellow triangle on LHS of screen */
        glColor4f(0.0, 0.0, 1.0, 0.75); //blue, alpha=0.75
        circle(1,0,2); //x=1,y=0,radius=2
        }
```

- Question:
  - If we change the sFactor and dFactor of glBlendFunc(sFactor, dFactor), what will be the blending effect?



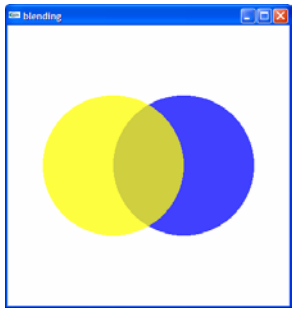


Init blending function in main

```
/* We will change sFactor and dFactor values later on to test the blending
   effect.*/

//(Sr,Sg,Sb,Sa)=(As,As,As,As)= (0.75,0.75,0.75,0.75)
sFactor = GL_SRC_ALPHA;

//(Dr,Dg,Db,Da)=(1-As,1-As,1-As,1-As)= (0.25,0.25,0.25,0.25)
dFactor = GL_ONE_MINUS_SRC_ALPHA;

/*  Initialize alpha blending function.  */
static void init(void)
{
   glEnable (GL_BLEND);
   glBlendFunc (sFactor, dFactor);
   glShadeModel (GL_FLAT);
   glClearColor (1, 1, 1, 1); // white and opaque background
}
```

Test Case A
Let sFactor = GL_SRC_ALPHA, dFactor = GL_ONE_MINUS_SRC_ALPHA
Draw blue circle first, yellow second.

```
void display() {
drawLeftCircle(); // yellow circle
drawRightCircle(); // blue circle
}
```



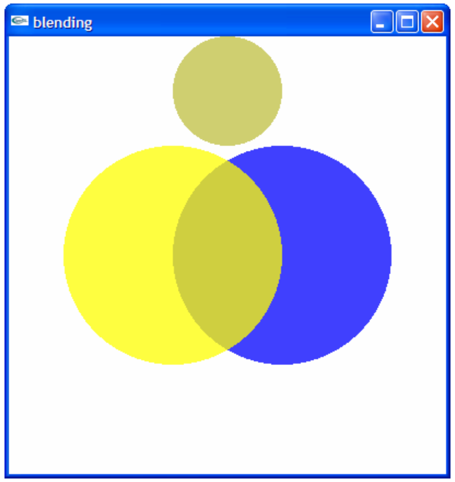
- Predicted final blending value based on formula:
  (Rf,Gf,Bf,Af) = (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)
  - (Rs,Gs,Bs,As) = (1,1,0,0.75) //yellow, semi-transparent
  - (Rd,Gd,Bd,Ad) = (0,0,1,0.75) //blue, semi-transparent
  - sFactor = GL_SRC_ALPHA
    - (Sr,Sg,Sb,Sa) = (0.75,0.75,0.75,0.75)
  - dFactor = GL_ONE_MINUS_SRC_ALPHA
    - (Dr,Dg,Db,Da) = (1-0.75,1-0.75,1-0.75,1-0.75) = (0.25,0.25,0.25,0.25)
- Final predicted blending value (Rf,Gf,Bf,Af) = (0.75,0.75,0.25,0.75)

Use our predicted final blending value to draw an object (circle) and compare its color with the blended color of the blue and yellow circles.

- In the preceding slide, our predicted final blending value was (Rf,Gf,Bf,Af) = (0.75,0.75,0.25,0.75)

- Function to draw a circle centered at (0,3,0) with radius of 1.

```
static void drawTestCircle(void)
{
glColor4f(0.75, 0.75, 0.25, 0.75);
circle(0,3,1);
}
```



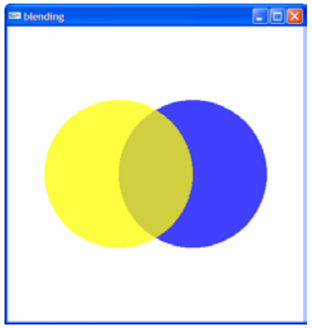
- Display the test circle together with two intersecting circles

```
void display() {
drawTestCircle();  // draw test circle with predicted color
drawRightCircle(); // Destination: blue circle
drawLeftCircle();  // Source: yellow circle
}
```

- Result:
  - The predicted color shown in the test circle matches the final blended color of the yellow and blue circles.

Test Case BLet sFactor = GL_SRC_ALPHA, dFactor = GL_ONE_MINUS_SRC_ALPHA
Draw yellow circle first, blue circle second.

```
void display() {
drawLeftCircle(); // yellow circle
drawRightCircle(); // blue circle
}
```



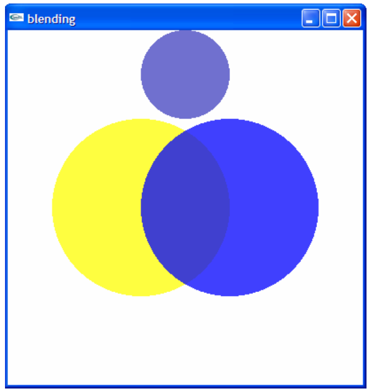
- Predicted final blending value based on formula:
  (Rf,Gf,Bf,Af) = (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)
  - (Rs,Gs,Bs,As) = (0,0,1,0.75) //blue, semi-transparent
  - (Rd,Gd,Bd,Ad) = (1,1,0,0.75) //yellow, semi-transparent
  - sFactor = GL_SRC_ALPHA
    - (Sr,Sg,Sb,Sa) = (0.75,0.75,0.75,0.75)
  - dFactor = GL_ONE_MINUS_SRC_ALPHA
    - (Dr,Dg,Db,Da) = (1-0.75,1-0.75,1-0.75,1-0.75) = (0.25,0.25,0.25,0.25)
- Final predicted blending value (Rf,Gf,Bf,Af) = (0.25,0.25,0.75,0.75)

Use our predicted final blending value to draw an object (circle) and compare the color with the blending color of the yellow and blue circles.

- In the preceding slide, our predicted final blending value was (Rf,Gf,Bf,Af) = (0.25,0.25,0.75,0.75)

- Function to draw a circle centered at (0,3,0) with radius of 1.

```
static void drawTestCircle(void)
{
glColor4f(0.25, 0.25, 0.75, 0.75);
circle(0,3,1);
}
```



- Display the test circle together with two intersecting circles

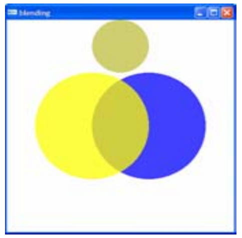
```
void display() {
drawTestCircle();  // draw test circle with predicted color
drawLeftCircle();  // Source: yellow circle
drawRightCircle(); // Destination: blue circle
}
```

- Result:
  - The predicted color shown in the test circle matches the final blending color of the yellow and blue circle.

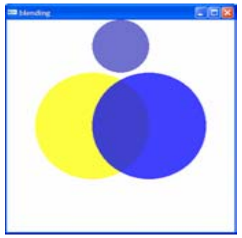
Conclusion drawn from Test A and Test B

- In both Test A and B, we use the same blending function:
  - glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
- The differences are:
  - In Test A, we draw first the blue, then the yellow circle.
  - In Test B, we draw first the yellow, then blue circle.
- Tests A and B display different blending effects.
- This shows that the order in which objects are defined makes a difference in the final color blending result.

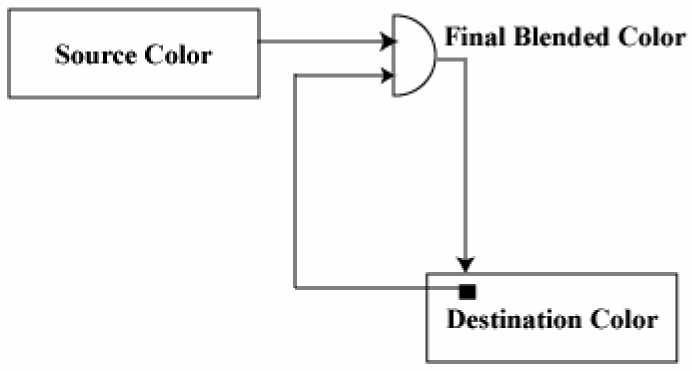
Test a



Test b



Blending Model



Case 3
Blending is dangerous and has many artifacts

Study Case:

- A cube emitting blue light.
  - One cube emitting blue light.

```
GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
```

- When light shines on the cube, the material reflects red diffuse color of alpha value 0.6.

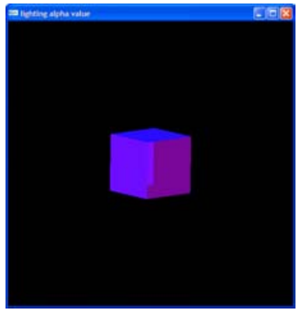
```
GLfloat mat_transparent1[] = { 1, 0, 0, 0.8}; //  red 0.8 diffuse
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent1);//red light
```

- One light shining in the –Z direction

```
static void init(void)
{
GLfloat position[] = { 0, 0, 1.0, 0 }; //light shining headon

glLightfv(GL_LIGHT0, GL_POSITION, position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
```

- Goal::
  - We will change the backgournd, the blending function to see what kind of artfact we can get. We will not be going through details as to what these artifacts are.
  - The purpose of this study case is to show how dangerous blending can be☺

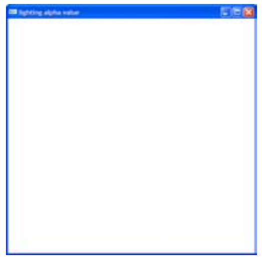
One of the artfacts



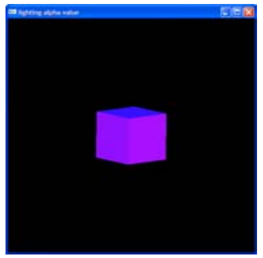
Case 3, A
Object is visible with black background but disappears with white background

```
void display(void) {
     GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 }; //black
     GLfloat mat_transparent[] = { 1, 0, 0, 0.6}; //red 0.6 diffuse
     GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue

     SetBackground(1,1,1,1);
     glRotatef (15.0, 1.0, 1.0, 0.0);
     glRotatef (30.0, 0.0, 1.0, 0.0);
     glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);//red 0.6 diffuse
     glEnable (GL_BLEND);
     glDepthMask (GL_FALSE);// make the depth buffer read only
                           // while drawing the translucent objects
     glBlendFunc (GL_SRC_ALPHA, GL_ONE);
    glCallList (cubeList);
    glDepthMask (GL_TRUE);
    glDisable (GL_BLEND);

     glFlush();
}
```

White background
SetBackground(1,1,1,1);



Black background
SetBackground(1,1,1,1);



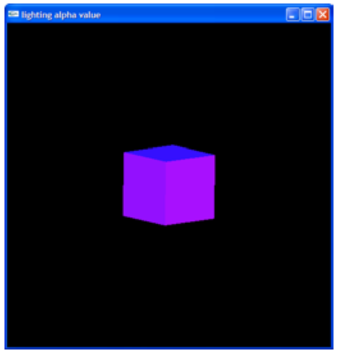
Note: This artifact can be noticed when using
glBlendFunc (GL_SRC_ALPHA, GL_ONE);

However, when using glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
it does not matter whether the background is black and white; the cube is always viewable.

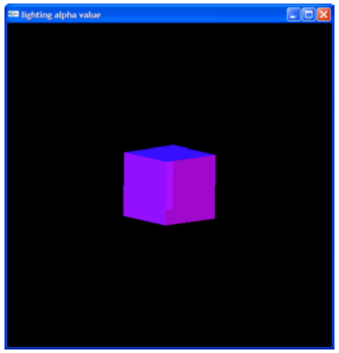
Case 3, B
Blending with depth buffer read-only vs depth buffer writable.

```
void display(void)
{
     GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 }; //black
     GLfloat mat_transparent[] = { 1, 0, 0, 0.6}; //red 0.6 diffuse
     GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue

     SetBackground(0,0,0,1);
     glRotatef (15.0, 1.0, 1.0, 0.0);
     glRotatef (30.0, 0.0, 1.0, 0.0);
     glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);//red 0.6 diffuse
     glEnable (GL_BLEND);
     glDepthMask (GL_FALSE);// make the depth buffer read only
                           // while drawing the translucent objects
     glBlendFunc (GL_SRC_ALPHA, GL_ONE);
    glCallList (cubeList);
    glDepthMask (GL_TRUE);
    glDisable (GL_BLEND);

     glFlush();
}
```

With depth buffer read-only
glDepthMask (GL_FALSE);



With depth buffer writable
glDepthMask (GL_TRUE);



Note: This artifact is noticed when using
glBlendFunc (GL_SRC_ALPHA, GL_ONE);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

However, combinations of all possible values for glBlendFunc were not attempted.

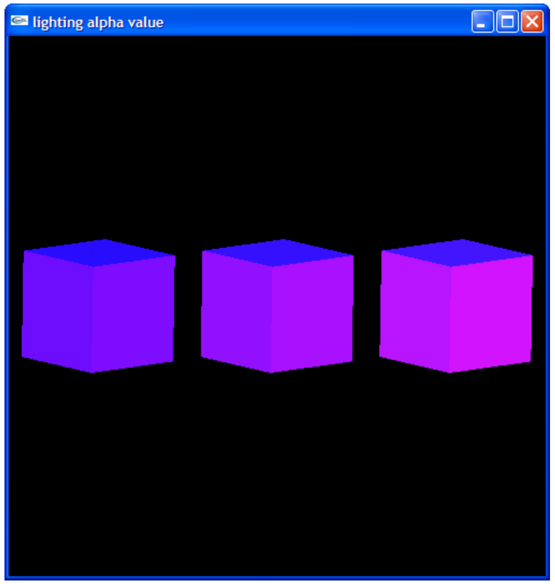
Lesson from Case 3

- Blending is dangerous with mixed with light whose alpha value is < 1 (transparent).

- In order to avoid artifacts, you need to know exactly what you are doing.

Case 4
How does the alpha value affect the object's color?

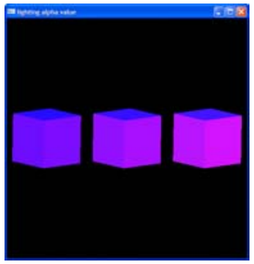
- Study Case:
  - There are three cubes emitting blue color.
  - If light is shine on the left, it will present red diffuse color of alpha value, such as 0.6 transparency.
  - If light is shined on the middle, it will present red diffuse color of alpha value, such as 0.8 (transparency).
  - If light is shine on the right, it will present red diffuse color of alpha = 1 (solid).
- Question:
  - How does the alpha value of light affect the color of the cube?



Case Study 4, continued …

Setup common traits for the three cubes to be rendered

- Set up light position (towards –Z direction)
- Set up material's specular property (dark grey), and shininess (very shiny)
  - Since our goal is not to study material's specular property, therefore, we minimize the specular color by setting it to dark grey (very negligible, and we set specular spot on the object by setting the material shininess to 127, very shiny.
- Create display list of the cube
  - This step is not required. However, since we will render the same cube three times to display three cubes, using display list will make the rendering faster.



```
static void init(void)
{
  //dark grey specular light
  GLfloat mat_specular[] = { 0.1, 0.1, 0.1, 1 }
  // concentrated specular spot
  GLfloat mat_shininess[] = { 127 };
  GLfloat position[] = { 0, 0, 1.0, 0 };

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  //light shining head on
  glLightfv(GL_LIGHT0, GL_POSITION, position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);  //one light source
  glEnable(GL_DEPTH_TEST);

  // create display list for a cube to be used later
  cubeList = glGenLists(1);
  glNewList(cubeList, GL_COMPILE);
  glutSolidCube (0.6);
  glEndList();
}
```

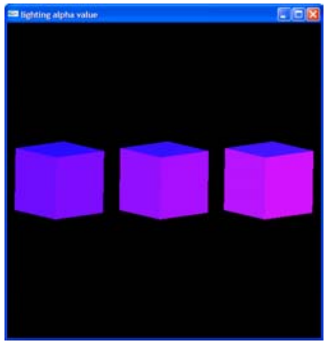
Case Study 4, continued …
Render three cubes with same emission color (solid blue) but different alpha value of red diffuse color

```
void display(void)
{
   GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 }; //black
   GLfloat mat_transparent1[] = { 1, 0, 0, 0.6}; //  red 0.6 diffuse
   GLfloat mat_transparent2[] = { 1, 0, 0, 0.8}; //  red 0.8 diffuse
   GLfloat mat_transparent3[] = { 1, 0, 0, 1};  //  red 1 diffuse
   GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue

   SetBackground(0,0,0,1); //black

    // Cube on the left: diffuse = red, alpha = 0.6
   glPushMatrix ();
   glTranslatef (-1, 0, 0);//
   glRotatef (15.0, 1.0, 1.0, 0.0);
   glRotatef (30.0, 0.0, 1.0, 0.0);
   glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent1);//red
             glEnable (GL_BLEND);
   glBlendFunc (GL_SRC_ALPHA, GL_ONE);
             glCallList (cubeList);
   glDisable (GL_BLEND);
   glPopMatrix ();

   // Cube in the middle: diffuse = red, alpha = 0.8
   glPushMatrix ();
   glTranslatef (0, 0, 0);
   glRotatef (15.0, 1.0, 1.0, 0.0); glRotatef (30.0, 0.0, 1.0, 0.0);
    glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent2);//red

    glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE);
    glCallList (cubeList);
    glDepthMask (GL_TRUE);
    glDisable (GL_BLEND);
   glPopMatrix ();

//Cube on the right: diffuse=red, alpha = 1
glPushMatrix ();
glTranslatef (1, 0, 0);
 glRotatef (15.0, 1.0, 1.0, 0.0);
glRotatef (30.0, 0.0, 1.0, 0.0);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
 glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent3);//red

glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE);
glCallList (cubeList);
glDisable (GL_BLEND);
glPopMatrix ();

glFlush();
}
```

Conclusion:
The alpha value of material affects the final object color.

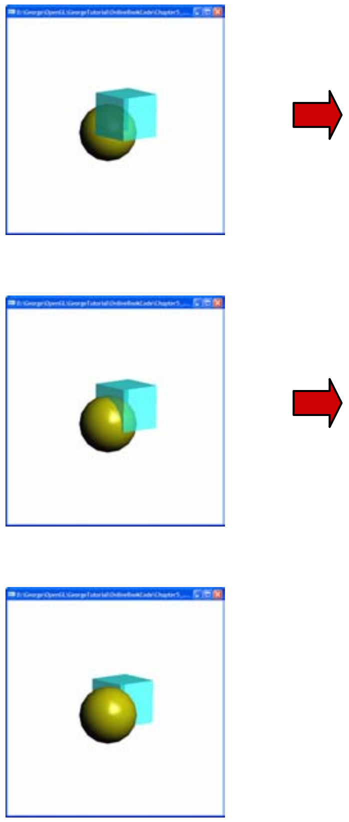
Between [0,1], the larger the alpha value, the Deeper the color.

In this case, the cube on the right has alpha value 1 for its diffuse color, therefore it diffuses more red. Since red (diffuse) + blue (emissive color) = purple, therefore, the cube on the right displays deeper purple.



Case Study 5
Blending with depth buffer

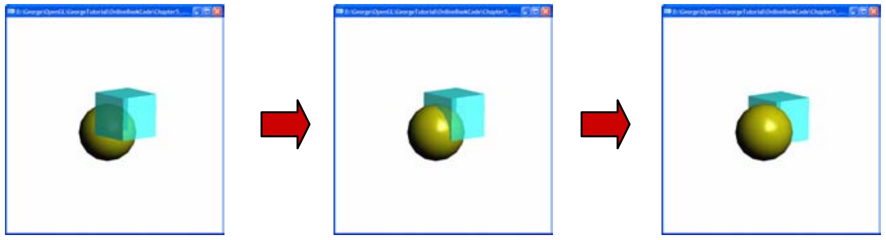
- Scenario:
  - In the scene, there is a translucent cyan cube in front of a solid yellowish sphere.
    - The cube is centered at (0.1, 0.1, 8)
    - The sphere is centered at (-0.1, -0.1, 8)
  - Each time user click “a” on the keyboard, the cube will move backwards a bit.
  - When user click “r” on the keyboard, the cube will be reset to its original position.
- Goal:
  - **The part of translucent cyan cube that is in front of the solid sphere blended with sphere.**
  - **The part of translucent cyan cube that is behind the sphere will not be displayed.**



The code to move the translucent cyan cube backward when user click “a”, and reset the position of cube, when user click “r”

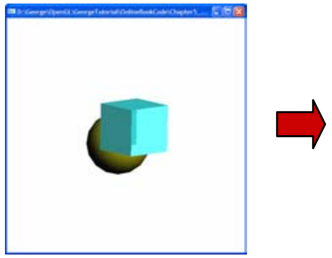
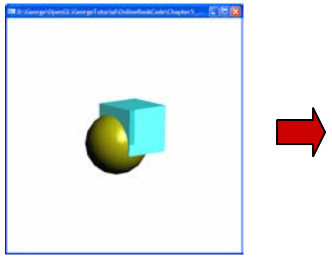
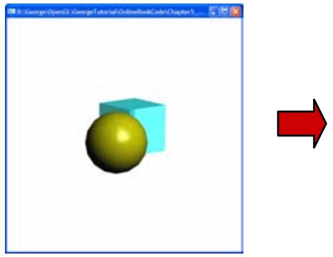
```
#define ZINC 1
#define MAXZ 8.0
#define MINZ -8.0
static float solidZ = MINZ;
static float transparentZ = MAXZ;

void ChangeLocation(void) {
     transparentZ -= ZINC;
        glutPostRedisplay();
}

void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 'a':
      case 'A':
     ChangeLocation();
         glutPostRedisplay();
         break;
      case 'r':
      case 'R':
     transparentZ = MAXZ;
         glutPostRedisplay();
         break;
      case 27:
        exit(0);
    }
}
```

- In the code left, when user click ‘a’ or ‘A”, the global valuable which keeps track of the location of the cyan cube is incremented.
- The result is that the cube is moved backward.
- As the transparent cube moves forward, **what we expect to see** is that it blends with the opaque sphere.



Our trouble, when the translucent cyan cube is in the front of solid sphere, the color of cube does not blend with the color of the sphere.

```
void display(void)
{
   …
   //glClear (GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT);
   SetBackground(1,1,1,1);

   // Draw the translucent cube farther from camera
   glPushMatrix ();
    glTranslatef (0.15, 0.15, transparentZ);
    …
    glEnable (GL_BLEND);

     glBlendFunc (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);

     glCallList (cubeList);
     glDisable (GL_BLEND);
   glPopMatrix ();

   // Draw the solid sphere closer to camera
   glPushMatrix ();
    glTranslatef (-0.15, -0.15, solidZ);
    …
    glCallList (sphereList); // draw the solid sphere
   glPopMatrix ();

   glutSwapBuffers();
}
```

How to fix the problem occurred when you draw both translucent and solid objects in one scene?

- It might be a bug in OpenGL that when you draw translucent objects mixed with solid objects in one scene, if draw the translucent object first, OpenGL treats it as if it were solid, blending will not occur.

- Fix:

  – If you want translucent objects (in front) blending with solid objects, and solid objects (in front) obscuring translucent objects, you need to exercise care if you draw the translucent and solid objects in one scene.

  – The way to draw the objects are as follows:

  (1) Enable the depth buffer.

  (2) Draw all opaque objects.

  (3) Enable blend, then draw all translucent objects.

New Code

Drawing the solid sphere before drawing the transparent cube.

```
void display(void){
   …
   //glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   SetBackground(1,1,1,1);

   // Draw the solid sphere closer to camera
   glPushMatrix ();
    glTranslatef (-0.15, -0.15, solidZ);
    …
    glCallList (sphereList); // draw the solid sphere
   glPopMatrix ();

 // Draw the translucent cube farther from camera
   glPushMatrix ();
    glTranslatef (0.15, 0.15, transparentZ);
    …
    glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);

    glCallList (cubeList);
    glDisable (GL_BLEND);

   glPopMatrix ();

   glutSwapBuffers();
}
```