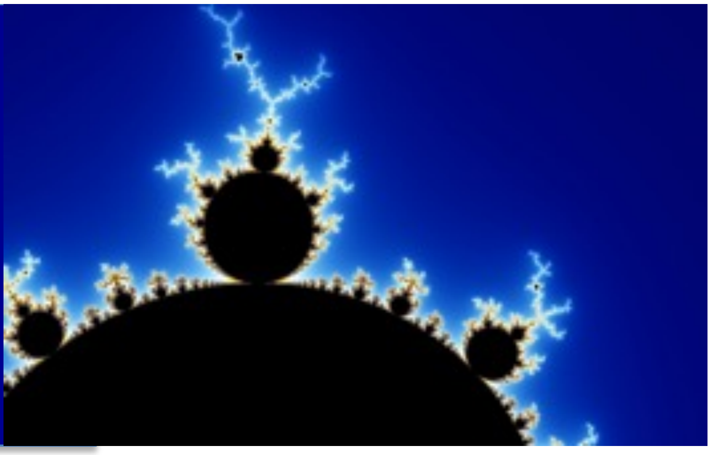
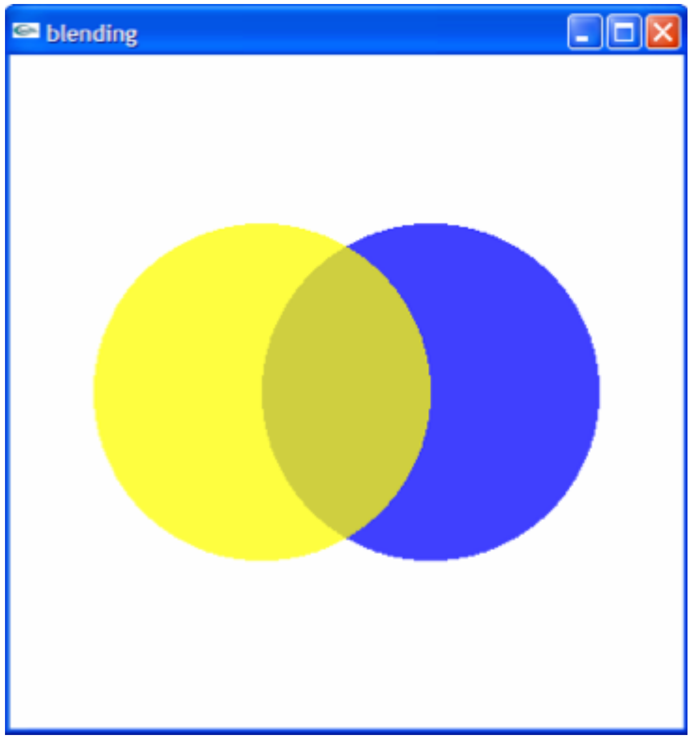
Computer Graphics

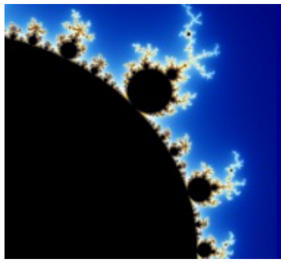


Transparency

How is transparency achieved in WebGL?

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



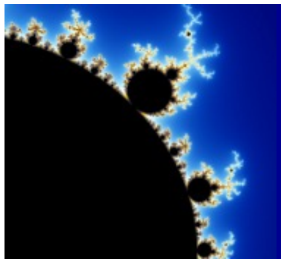


Enable/Disable blending of color

- To enable transparency, we need to explicitly enable blending:
  - gl.enable(gl.BLEND);   // blend is off by default
  - gl.disable(gl.DEPTH_TEST); // have to disable depth test, otherwise, inconsistent blending results
- To disable blending:
  - gl.disable(gl.BLEND)
- Alpha value

0 <= alpha <= 1

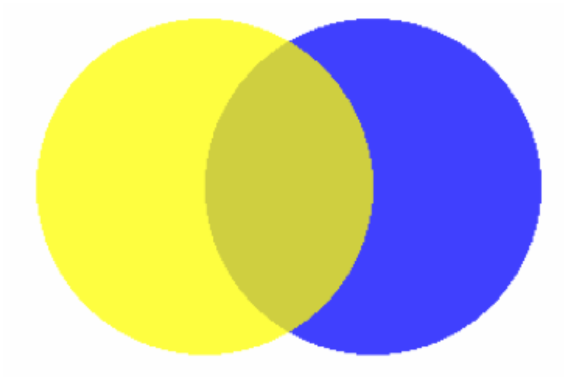
Transparent opaque



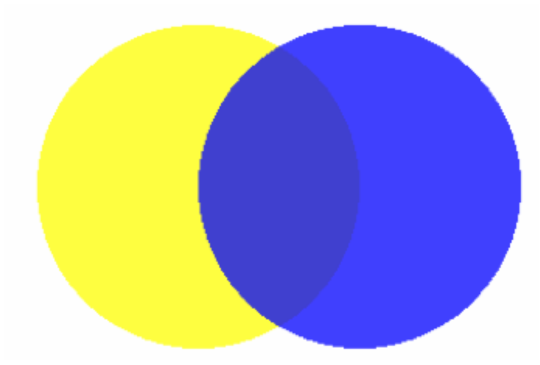
Case Study 1

- Scenario:
  - There's a yellow circle and a blue circle. The yellow circle is 75% opaque (alpha=0.75). The blue circle is 25% opaque (alpha=0.25).
- Question:
  - When the yellow circle is in front of the blue circle, i.e., blue circle is drawn before yellow circle, what do we see, Choice A or Choice B?

- Choice A



- Choice B



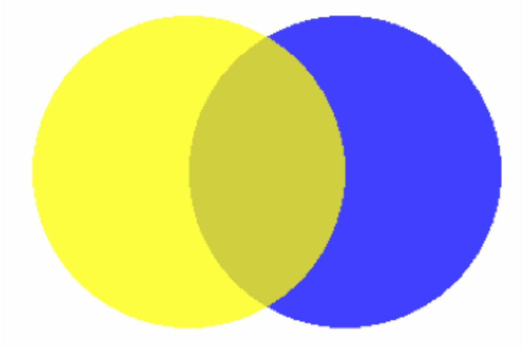
Case Study 1 – Answer is A

- **Reason:** Because yellow circle is more opaque, therefore we see more of the yellow and less of the blue.

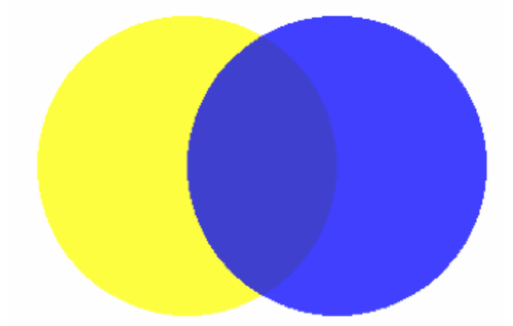
- Scenario:
  - the yellow circle is 75% opaque(alpha=0.75) and blue circle is 25% opaque(alpha=0.25).

- The yellow circle is in front of the blue circle, only 25% of the blue blends with 75% of the yellow. Result is Choice A.

- Choice A



- Choice B



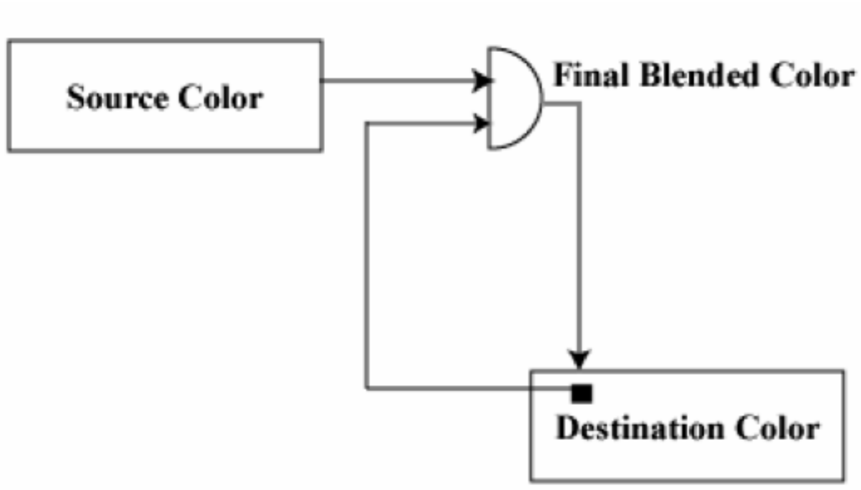
What happens when blending is enabled?

- 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 frame buffer.
  - Blending occurs after the scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the frame buffer.

- **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.

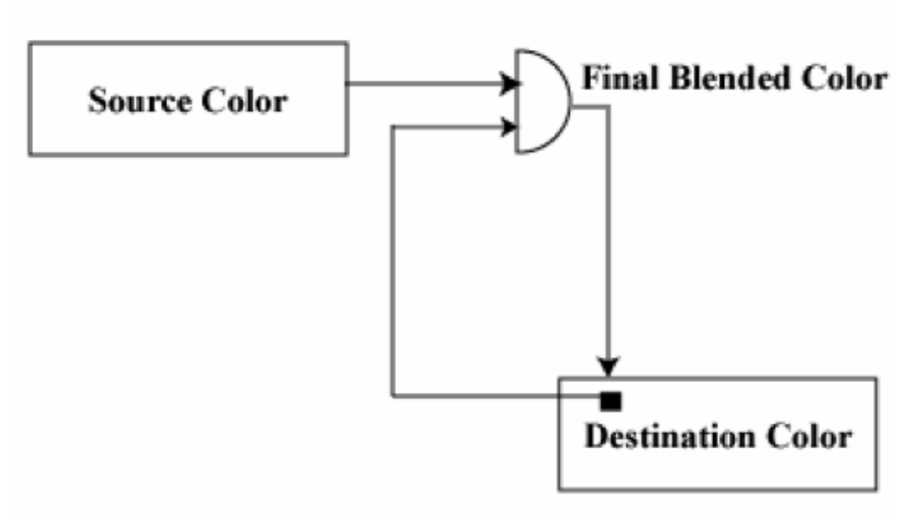
Blending Model

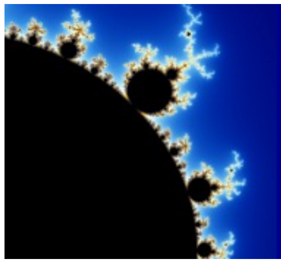


The source and destination factors

- During blending, source is the color value of the incoming fragment.
- Destination is the currently stored pixel value.

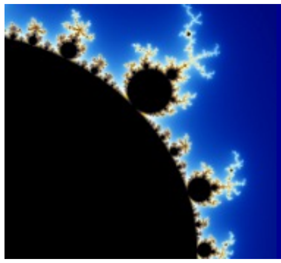
Blending Model





How does WebGL blend the source and destination color values?

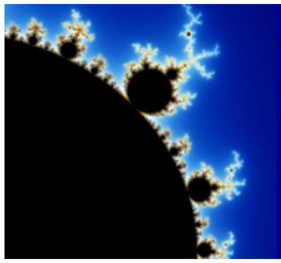
- Assume: Source Color: (Rs, Gs, Bs, As)
  Destination Color: (Rd, Gd, Bd, Ad)
- Step 1:
  - Specify the source and destination **blending factors**.
    - Let source destination blending factors be: – (Sr, Sg, Sb, Sa)
    - Let destination blending factors be: – (Dr, Dg, Db, Da)
- Step 2:
  - Combine 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 and Blend Factors

void **gl.blendFunc**(GLenum *sfactor*, GLenum *dfactor*)

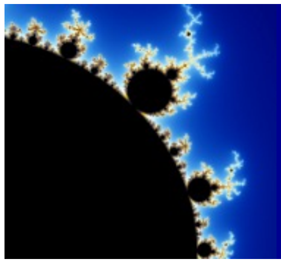
- *sfactor: * Specifies how the red, green, blue and alpha **source blending factors** are computed. 9 symbolic constants are accepted:
  - **gl.ZERO**
  - **gl.ONE**
  - **gl.DST_COLOR**
  - **gl.ONE_MINUS_DST_COLOR**
  - **gl.SRC_ALPHA**
  - **gl.ONE_MINUS_SRC_COLOR**
  - **gl.DST_ALPHA**
  - **gl.ONE_MINUS_DST_ALPHA**
  - **gl.SRC_ALPHA_SATURATE**
- *dfactor:* Specifies how the red, green, blue and alpha **destination blending factors** are computed. 8 symbolic constants are accepted:
  - **gl.ZERO**
  - **gl.ONE**
  - **gl.SCR_COLOR**
  - **gl.ONE_MINUS_SRC_COLOR**
  - **gl.SRC_ALPHA**
  - **gl.ONE_MINUS_SRC_ALPHA**
  - **gl.DST_ALPHA**
  - **gl.ONE_MINUS_DST_ALPHA.**



Source and Destination Blending Factors Table

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	(Rd, Gd, Bd, Ad)
gl.SRC_COLOR	Destination	(Rs, Gs, Bs, As)
gl.ONE_MINUS_SRC_COLOR	DESTINATION	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
gl.ONE_MINUS_DST_COLOR	SOURCE	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
gl.SRC_ALPHA	Source or destination	(As, As, As, As)
gl.ONE_MINUS_SRC_ALPHA	Source or destination	(1, 1, 1, 1) – (As, As, As, As)
gl.DST_ALPHA	Source or destination	(Ad Ad, Ad, Ad)
gl.ONE_MINUS_DST_ALPHA	Source or destination	(1, 1, 1, 1) – (Ad, Ad, Ad, Ad)
gl.SRC_ALPHA_SATURATE	Source	(f, f, f, 1); f=min(As, 1-Ad)

Source and destination factors may not both reference constant color



Blend Function

void **gl.blendFunc**(GLenum *sfactor*, GLenum *dfactor*)

- Used when the RGB color is blended the same way as the Alpha value
  - Pre-multiplied destination color, e.g., texture from images
- When using pure RBG colors without pre-multiplied alpha values:
  - The Alpha value should not be average/weighted the same way the RBG colors.
  - A different blending function API should be used

Blend Function

- void **gl.blendFuncSeperate**(

  GLenum *srcRGB*, GLenum *dstRGB,*
  Glenum *srcAlpha,*Glenum *destAlpha*
)
- Default values :
  - srcRGB(gl.ONE), dstRBG(gl.ZERO), srcAlpha(gl.ONE), dstAlpha(gl.ZERO)

- Experiment with different RBG and Alpha functions
  - Interactive Tool for understanding gl.blendFuncSeperate

Case Study 2: Source and Destination Blending Factors

- **Case Study Setup:**
  - **A yellow circle of radius .5, centered at (-0.3,0,0)**

    /* draw yellow circle on LHS of screen */

    color: (1.0, 1.0, 0.0, 0.75); //yellow, alpha=0.75

  - **A blue circle of radius .5, centered at (0.3,0,0)**

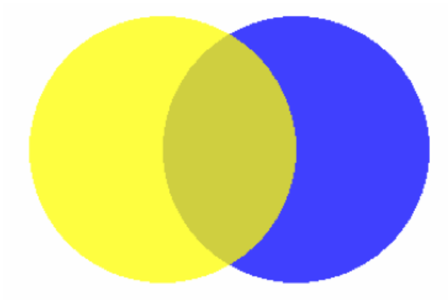
    /* draw blue circle on RHS of screen */

    color (0.0, 0.0, 1.0, 0.5); //blue, alpha=0.5

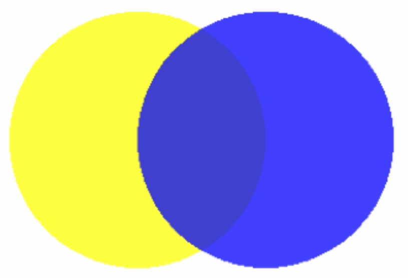
Question:

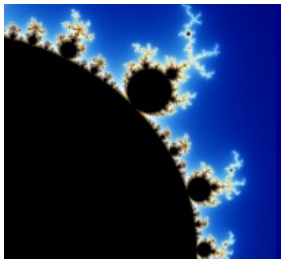
- What is the destination color ? What is the value for gl.SRC_ALPHA?
  - Depend on which circle is drawn first, and which circle is drawn next

- Choice A



- Choice B





Example Problems

gl.blendFuncSeperate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ONE, gl.ZERO)

– **A yellow circle of radius .5, centered at (-0.3,0,0)**

```
/* draw yellow circle on LHS of screen */
color: (1.0, 1.0, 0.0, 0.75); //yellow, alpha=0.75
```

– **A blue circle of radius .5, centered at (0.3,0,0)**

```
/* draw blue circle on RHS of screen */
color (0.0, 0.0, 1.0, 0.5); //blue, alpha=0.5
```

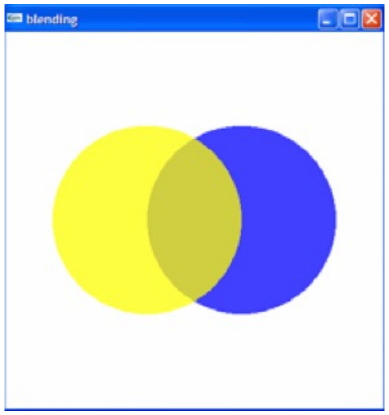
Test Case A:

drawRightCircle(); // draw blue circle first → **destination**
drawLeftCircle(); // yellow circle next → **source**

RGB :
Source blending factor : (Sr,Sg,Sb)=(As,As,As)= (0.75,0.75,0.75)
Destination blending factor : (Dr,Dg,Db)=(1-As,1-As,1-As)= (0.25,0.25,0.25)

Alpha :
Source blending factor : (Sa) = 1,
Destination blending factor : (Da) = 0

Test Case A : Let srcRGB = gl.SRC_ALPHA, dstRGB = gl.ONE_MINUS_SRC_ALPHA, srcAlpha=gl.ONE, dstAlpha=gl.ZERO



- Predicted final blending value based on formula:

*(Rf,Gf,Bf) = (Rs*Sr+Rd*Dr, Gs*Sg+Gd*Dg, Bs*Sb+Bd*Db)*

	source	destination
color	(Rs,Gs,Bs) (1,1,0) yellow	(Rd,Gd,Bd) (0,0,1) blue
Blending factor	gl.SRC_ALPHA (Sr,Sg,Sb) (0.75,0.75,0.75)	gl.ONE_MINUS_SRC_ALPHA (Dr,Dg,Db) (1-0.75,1-0.75,1-0.75) = (0.25,0.25,0.25)
Blended RGB color	(1*0.75+0*0.25, 1*0.75+0*0.25, 0*0.75+1*0.25) = (0.75,0.75,0.25)	

*(Af) = As*Sa + Ad*Da* ((Sa) = 1, (Da) = 0)

*Blended alpha = 0.75*1 + 0.5*0 = 0.75*

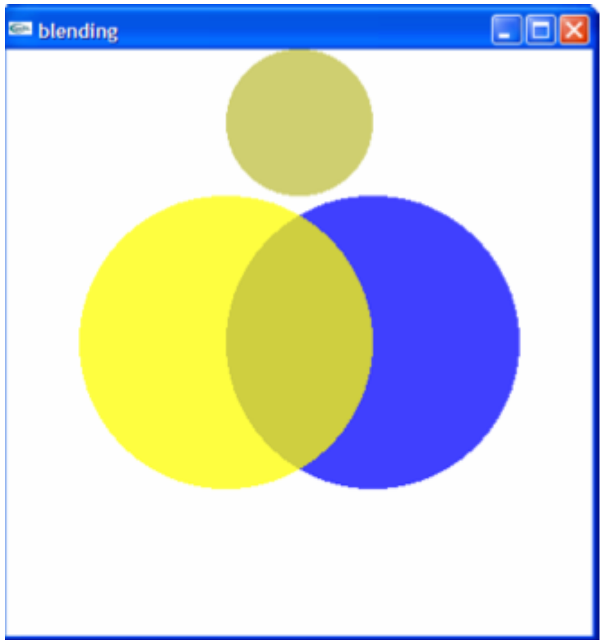
Use the predicted final blending value to draw an object (circle) and compare its color with the blended color of the blue and yellow circles

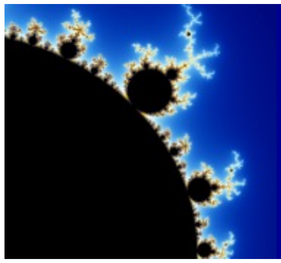
- In the preceding slide, our predicted final blending value was (Rf,Gf,Bf,Af) = (0.75,0.75,0.25,0.75)
- Function to draw a circle centered at

  (-0.3,0) with radius of 0.2. Display the test circle together with two intersecting circles

  Result:

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





Test Case B : Let srcRBG = gl.SRC_ALPHA, dstRGB = gl.ONE_MINUS_SRC_ALPHA, srcAlpha=gl.ONE, dstAlpha=gl.ZERO

drawLeftCircle(); // draw yellow circle first → **destination**

drawRightCircle(); // blue circle → **source**

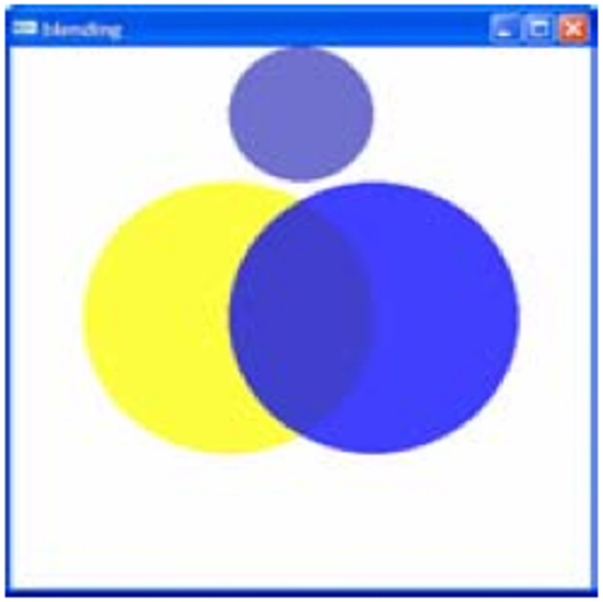
	source	**destination**
color	(Rs,Gs,Bs) (0,0,1) blue	(Rd,Gd,Bd) (1,1,0) yellow
Blending factor	gl.SRC_ALPHA (Sr,Sg,Sb) (0.5,0.5,0.5)	gl.ONE_MINUS_SRC_ALPHA (Dr,Dg,Db) (1-0.5,1-0.5,1-0.5) = (0.5,0.5,0.5)
Blended RBG color	(0*0.5+1*0.5, 0*0.5+1*0.5, 1*0.5+0*0.5) = (0.5,0.5,0.5)	

*(Af) = As*Sa + Ad*Da* ((Sa) = 1, (Da) = 0)

*Blended alpha = 0.5*1 + 0.75*0 = 0.5*

Use the 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) = (0.5,0.5,0.5,0.5)



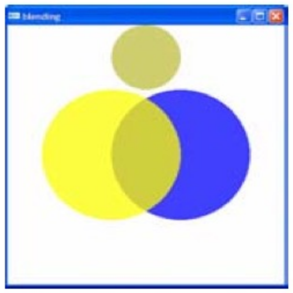
Conclusion drawn from Test A and Test B

- In both TestA and B, the same blending function was used:

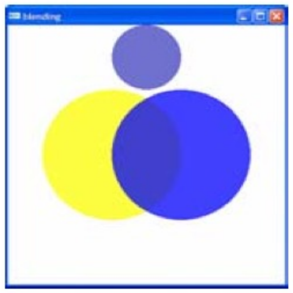
  gl.blendFuncSeperate(gl.SRC_ALPHA,
  gl.ONE_MINUS_SRC_ALPHA,
  gl.ONE, gl.ZERO)

- The differences are:
  - In Test A, first draw the blue, then draw the yellow circle.
  - In Test B, first draw the yellow, then draw the 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.

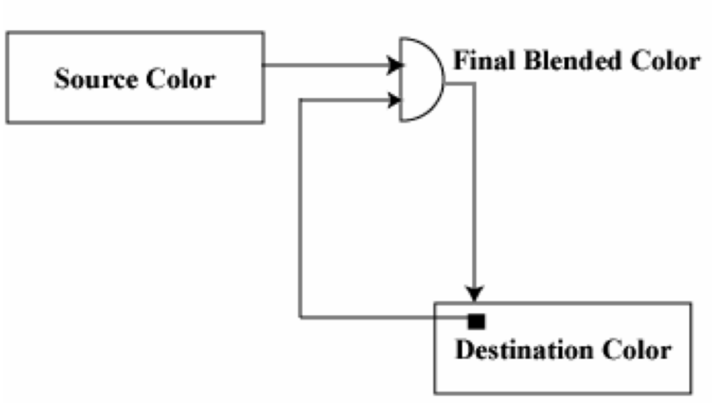
Test a

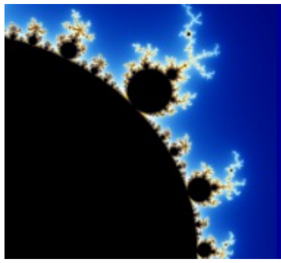


Test b



Blending Model





Practice Question

Two circles are drawn, the circle with color (0.5, 0.8, 1.0, 0.8) is drawn first and the circle with color (1, 0, 1, 0.5) is drawn next.

gl.blendFuncSeperate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ONE, gl.ZERO)

is used for color blending. Compute the color of the part of figure where the two circles overlap.

Practice Question

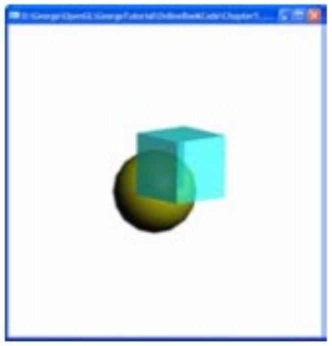
Two circles are drawn, the first circle drawn has color (0.5, 0.8, 1.0, 0.8) and the second circle drawn has color (1, 0, 1, 0.5),

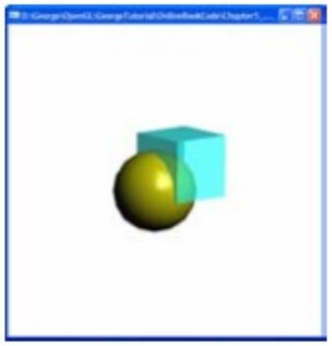
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA)

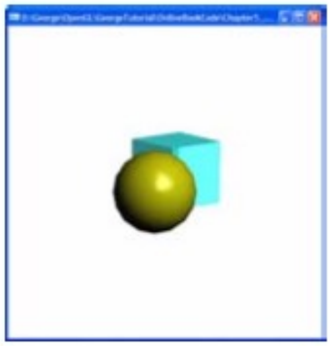
is used for color blending. Compute the color of the part of figure where the two circles overlap.

Case Study 4: Blending with depth buffer

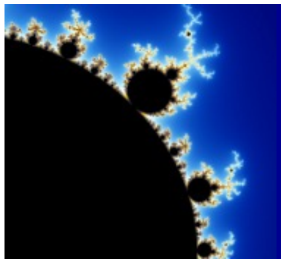
- Scenario:
  - In the scene, there is a translucent cyan cube in front of a solid yellowish sphere.
    - The cube is centered at (0.1, 0.1, 8)
    - The sphere is centered at (-0.1, -0.1, 8)
  - Each time user click “a” on the keyboard, the cube will move backwards a bit.
  - When user click “r” on 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.**



Blending with depth buffer

- 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.