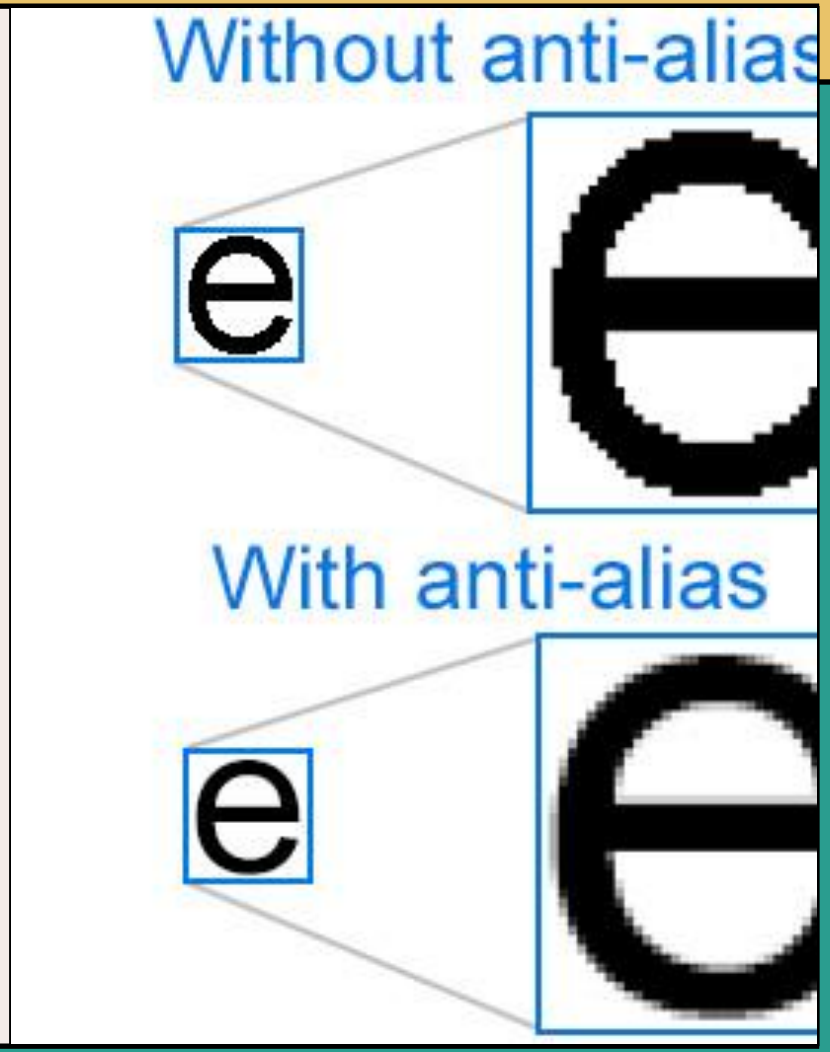
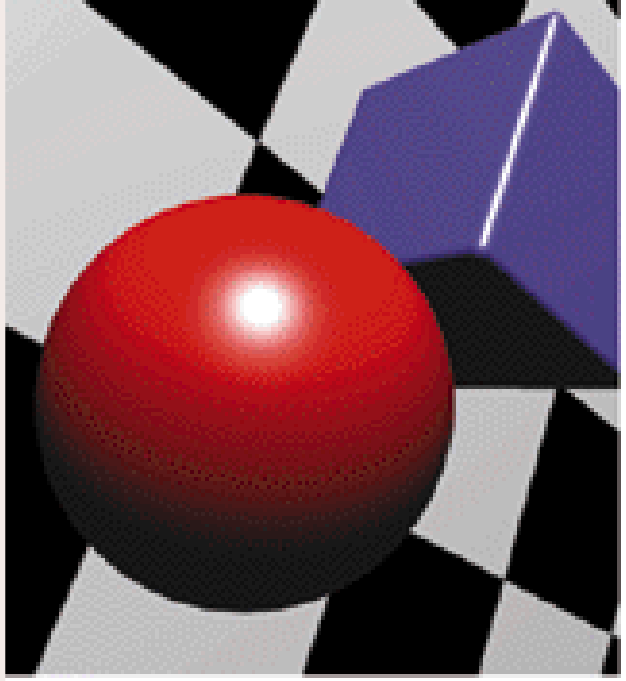
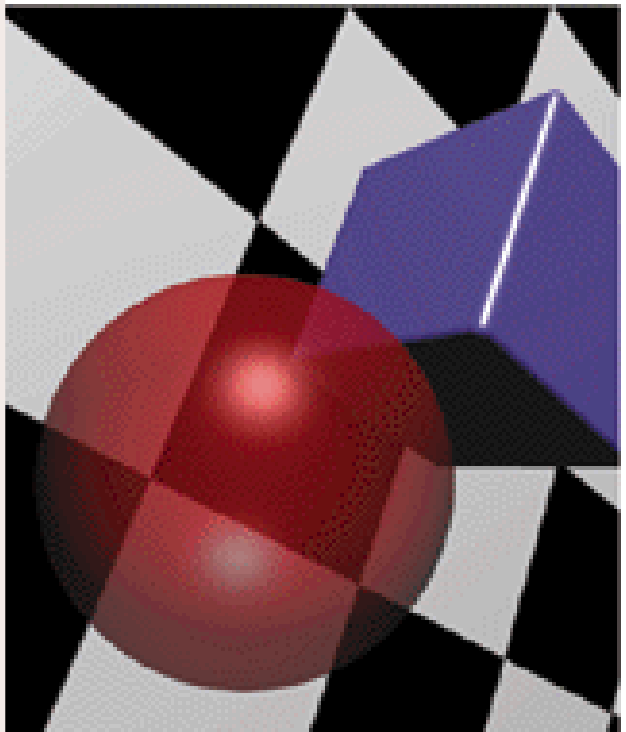
Blending and Fog

Mark Daniel Dacer







Blending

You've already seen alpha values (alpha is the A in RGBA), but they've been ignored until now.

Blending

- When blending is enabled, the alpha value is often used to combine the color value of the fragment being processed with that of the pixel already stored in the framebuffer. Blending occurs after your scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the framebuffer. Alpha values can also be used in the alpha test to accept or reject a fragment based on its alpha value.
- Without blending, each new fragment overwrites any existing color values in the framebuffer, as though the fragment were opaque. With blending, you can control how (and how much of) the existing color value should be combined with the new fragment's value. Thus you can use alpha blending to create a translucent fragment that lets some of the previously stored color value "show through." Color blending lies at the heart of techniques such as transparency, digital compositing, and painting.

Blending (cont.)

- The most natural way to think of blending operations is to think of the RGB components of a fragment as representing its color and the alpha component as representing opacity. Transparent or translucent surfaces have lower opacity than opaque ones and, therefore, lower alpha values. For example, if you're viewing an object through green glass, the color you see is partly green from the glass and partly the color of the object. The percentage varies depending on the transmission properties of the glass: If the glass transmits 80 percent of the light that strikes it (that is, has an opacity of 20 percent), the color you see is a combination of 20 percent glass color and 80 percent of the color of the object behind it. You can easily imagine situations with multiple translucent surfaces. If you look at an automobile, for instance, its interior has one piece of glass between it and your viewpoint; some objects behind the automobile are visible through two pieces of glass.

The Source and Destination Factors

- During blending, color values of the incoming fragment (the source) are combined with the color values of the corresponding currently stored pixel (the destination) in a two-stage process. First you specify how to compute source and destination factors. These factors are RGBA quadruplets that are multiplied by each component of the R, G, B, and A values in the source and destination, respectively. Then the corresponding components in the two sets of RGBA quadruplets are added. To show this mathematically, let the source and destination blending factors be (Sr, Sg, Sb, Sa) and (Dr, Dg, Db, Da), respectively, and the RGBA values of the source and destination be indicated with a subscript of s or d. Then the final, blended RGBA values are given by
- (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)
- Each component of this quadruplet is eventually clamped to [0,1].

The Source and Destination Factors (cont.)

- Now consider how the source and destination blending factors are generated. You use **glBlendFunc()** to supply two constants: one that specifies how the source factor should be computed and one that indicates how the destination factor should be computed. To have blending take effect, you also need to enable it:
  - **glEnable(GL_BLEND);**
- Use **glDisable()** with **GL_BLEND** to disable blending. Also note that using the constants **GL_ONE** (source) **and GL_ZERO** (destination) gives the same results as when blending is disabled; these values are the default.

The Source and Destination Factors (cont.)

- void glBlendFunc(*GLenum* ***sfactor****, GLenum* ***dfactor***);
- Controls how color values in the fragment being processed (the source) are combined with those already stored in the framebuffer (the destination). The argument sfactor indicates how to compute a source blending factor; dfactor indicates how to compute a destination blending factor. The possible values for these arguments are explained in Table 6-1. The blend factors are assumed to 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].

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_DST_COLOR	source	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
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)

Sample Uses of Blending

- Not all combinations of source and destination factors make sense. Most applications use a small number of combinations. The following paragraphs describe typical uses for particular combinations of source and destination factors. Some of these examples use only the incoming alpha value, so they work even when alpha values aren't stored in the framebuffer. Also note that often there's more than one way to achieve some of these effects.

- One way to draw a picture composed half of one image and half of another, equally blended, is to set the source factor to GL_ONE and the destination factor to GL_ZERO, and draw the first image. Then set the source factor to GL_SRC_ALPHA and destination factor to GL_ONE_MINUS_SRC_ALPHA, and draw the second image with alpha equal to 0.5. This pair of factors probably represents the most commonly used blending operation. If the picture is supposed to be blended with 0.75 of the first image and 0.25 of the second, draw the first image as before, and draw the second with an alpha of 0.25.
- To blend three different images equally, set the destination factor to GL_ONE and the source factor to GL_SRC_ALPHA. Draw each of the images with an alpha equal to 0.3333333. With this technique, each image is only one-third of its original brightness, which is noticeable where the images don't overlap.
- The blending functions that use the source or destination colors - GL_DST_COLOR or GL_ONE_MINUS_DST_COLOR for the source factor and GL_SRC_COLOR or GL_ONE_MINUS_SRC_COLOR for the destination factor - effectively allow you to modulate each color component individually. This operation is equivalent to applying a simple filter - for example, multiplying the red component by 80 percent, the green component by 40 percent, and the blue component by 72 percent would simulate viewing the scene through a photographic filter that blocks 20 percent of red light, 60 percent of green, and 28 percent of blue.

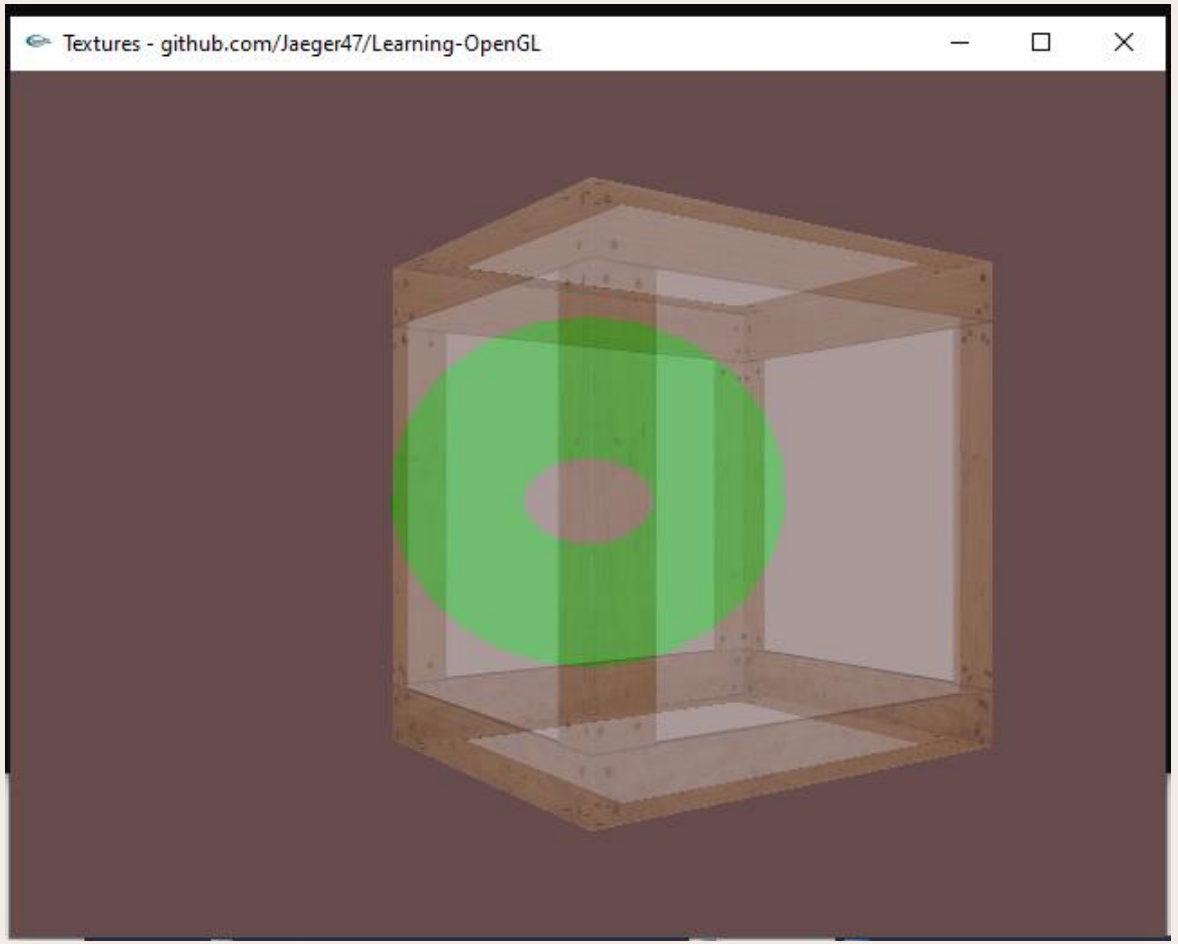
Preview

- Blending Setup

```
glEnable(GL_AUTO_NORMAL);
glEnable(GL_COLOR_MATERIAL);
glShadeModel(GL_SMOOTH);
glEnable (GL_BLEND); //enable blening fucntions
glDepthMask(GL_FALSE); //to avoid overlapping draw
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

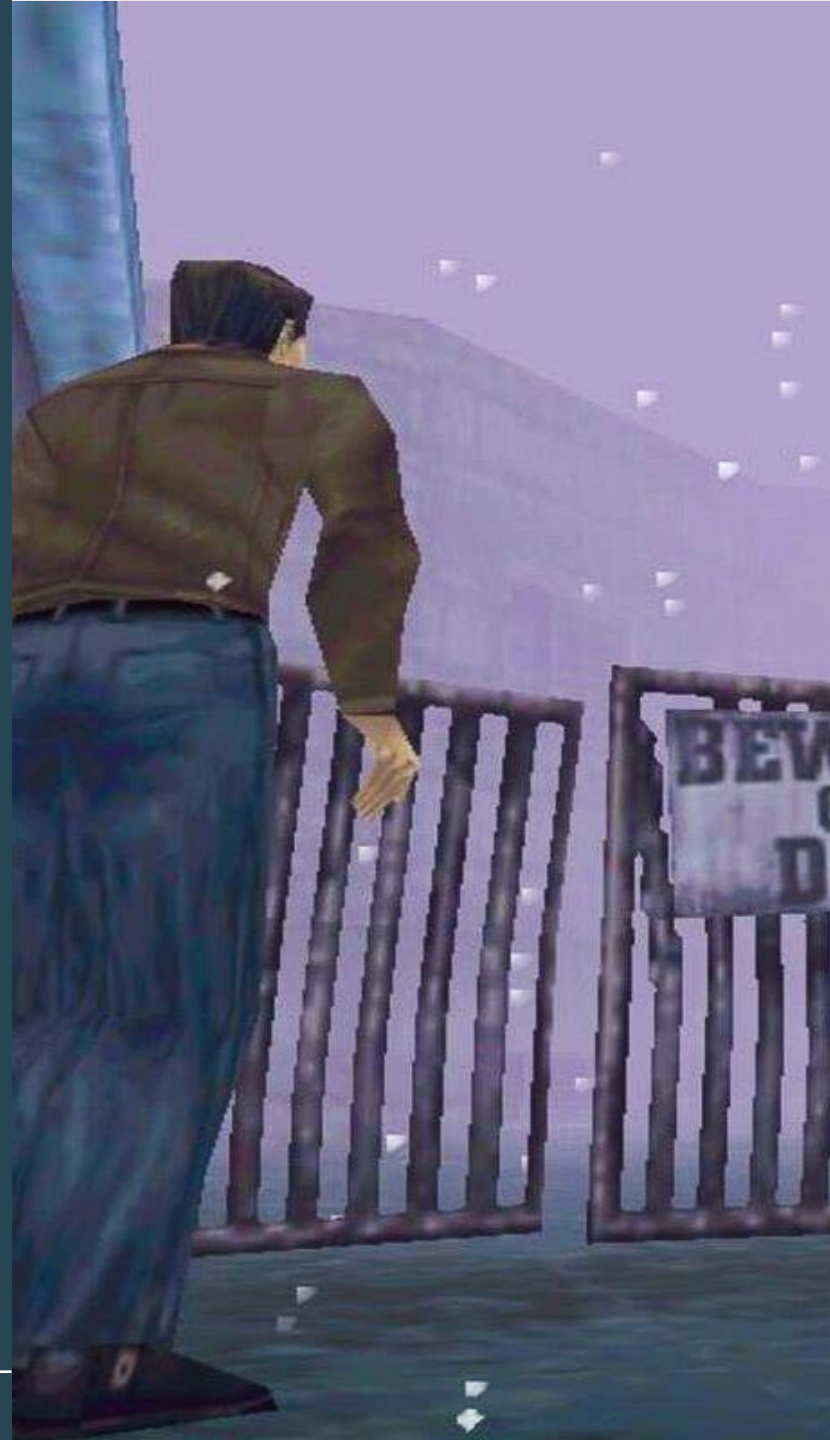
- Assigned 0.25 alpha to the box

```
glColor4f(1.0f,1.0f,1.0f,0.25);//
//front
glNormal3f(0.0f, 0.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); //textu
glVertex3f(-1.0f, -1.0f, 0.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(1.0f, -1.0f, 0.0f);
```



Fog

Computer images sometimes seem unrealistically sharp and well defined. Antialiasing makes an object appear more realistic by smoothing its edges.



Fog

- Computer images sometimes seem unrealistically sharp and well defined. Antialiasing makes an object appear more realistic by smoothing its edges. Additionally, you can make an entire image appear more natural by adding fog, which makes objects fade into the distance. Fog is a general term that describes similar forms of atmospheric effects; it can be used to simulate haze, mist, smoke, or pollution. (See Plate 9.) Fog is essential in visual-simulation applications, where limited visibility needs to be approximated. It's often incorporated into flight-simulator displays.

Fog (Cont.)

- When fog is enabled, objects that are farther from the viewpoint begin to fade into the fog color. You can control the density of the fog, which determines the rate at which objects fade as the distance increases, as well as the fog's color. Fog is available in both RGBA and color-index modes, although the calculations are slightly different in the two modes. Since fog is applied after matrix transformations, lighting, and texturing are performed, it affects transformed, lit, and textured objects. Note that with large simulation programs, fog can improve performance, since you can choose not to draw objects that would be too fogged to be visible.

Using Fog

- Using fog is easy. You enable it by passing GL_FOG to glEnable(), and you choose the color and the equation that controls the density with glFog*(). If you want, you can supply a value for GL_FOG_HINT with glHint()
- Fog Equations
  - Fog blends a fog color with an incoming fragment's color using a fog blending factor. This factor, f, is computed with one of these three equations and then clamped to the range [0,1].

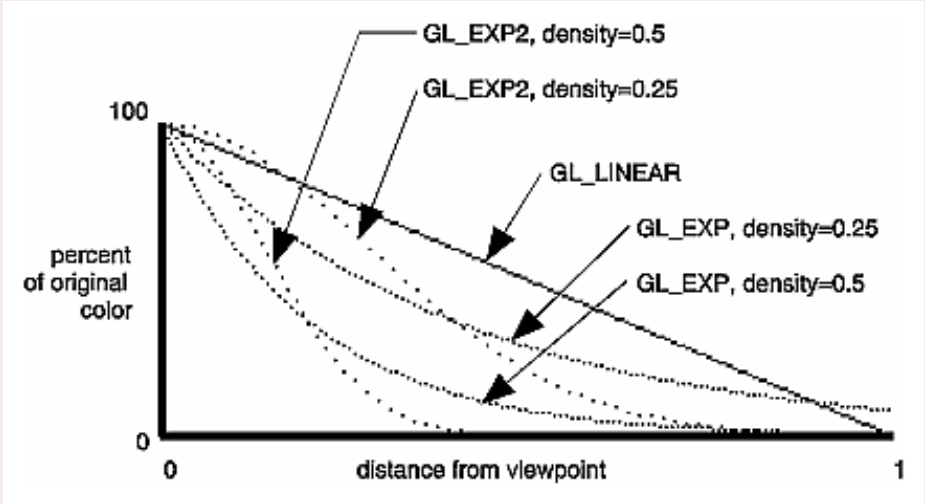
$$f = e^{-(density \cdot z)} \text{ (GL\_EXP)}$$

$$f = e^{-(density \cdot z)^2} \text{ (GL\_EXP2)}$$

$$f = \frac{end - z}{end - start} \text{ (GL\_LINEAR)}$$

Using Fog(cont.)

- **void glFog{if}(GLenum pname, TYPE param);**
- **void glFog{if}v(GLenum pname, TYPE *params);**
  - Sets the parameters and function for calculating fog. If pname is GL_FOG_MODE, then param is either GL_EXP (the default), GL_EXP2, or GL_LINEAR to select one of the three fog factors. If pname is GL_FOG_DENSITY, GL_FOG_START, or GL_FOG_END, then param is (or points to, with the vector version of the command) a value for density, start, or end in the equations. (The default values are 1, 0, and 1, respectively.) In RGBA mode, pname can be GL_FOG_COLOR, in which case params points to four values that specify the fog's RGBA color values. The corresponding value for pname in color-index mode is GL_FOG_INDEX, for which param is a single value specifying the fog's color index.



Preview

- Fog Setup

```
glEnable(GL_FOG);
{
  GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};

  fogMode = GL_EXP; //fog mode algo
  glFogi (GL_FOG_MODE, fogMode);
  glFogfv (GL_FOG_COLOR, fogColor);
  glFogf (GL_FOG_DENSITY, 0.03);  //density o
  glHint (GL_FOG_HINT, GL_NICEST);
  //fog distance
  glFogf (GL_FOG_START, 1.0);
  glFogf (GL_FOG_END, 5.0);
}
```