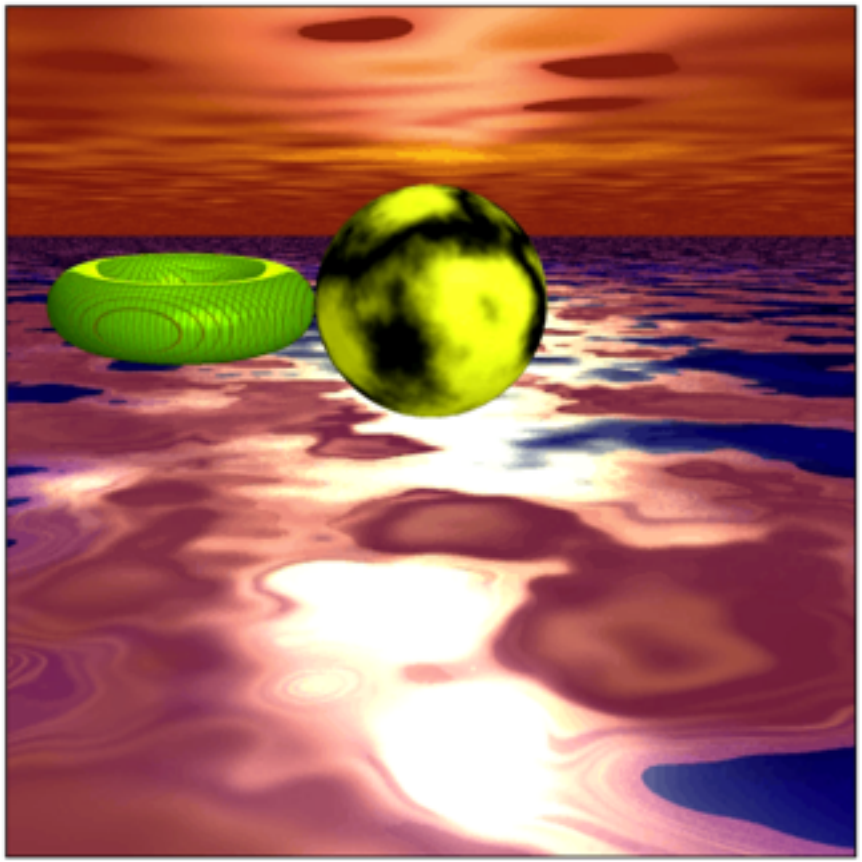
Supplement to Lecture 20

Image Composition



Alpha Channel

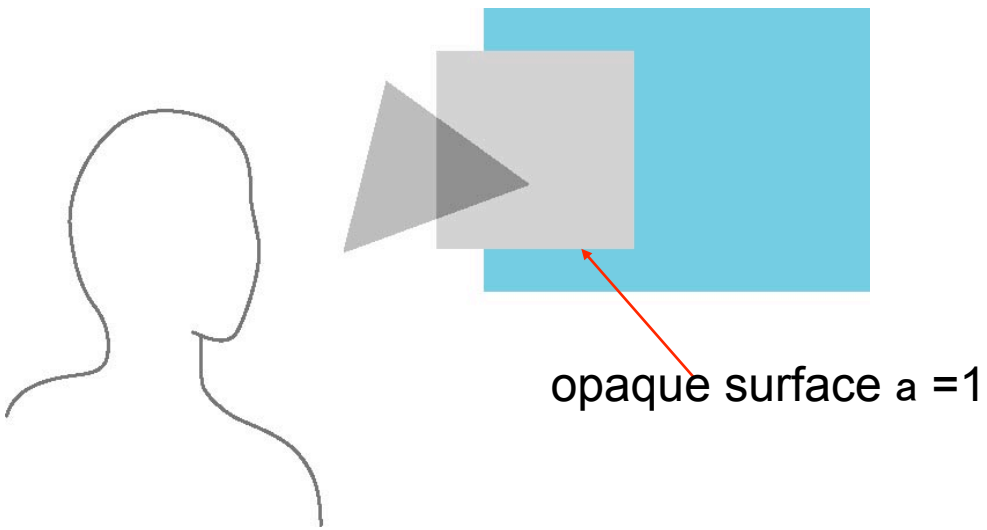
- Learn to use the A(lpha) component in RGBA color for
  - Blending for translucent surfaces
  - Compositing images

  - Antialiasing

Opacity and Transparency

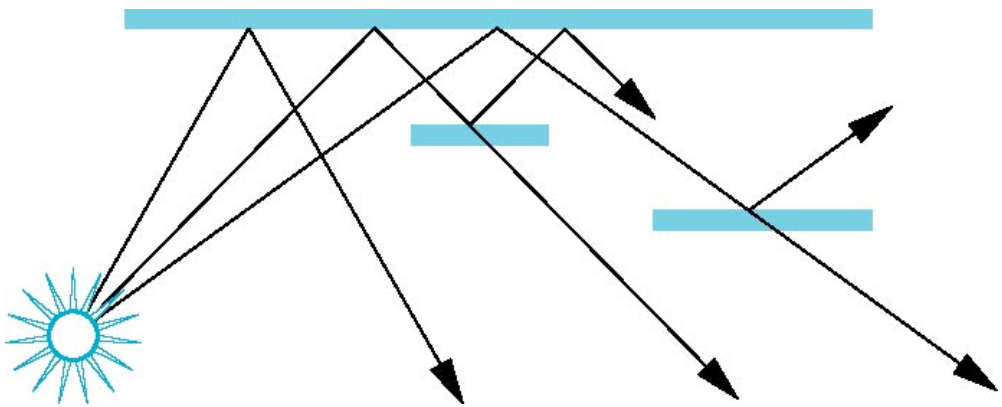
- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

  translucency = 1 – opacity (a)



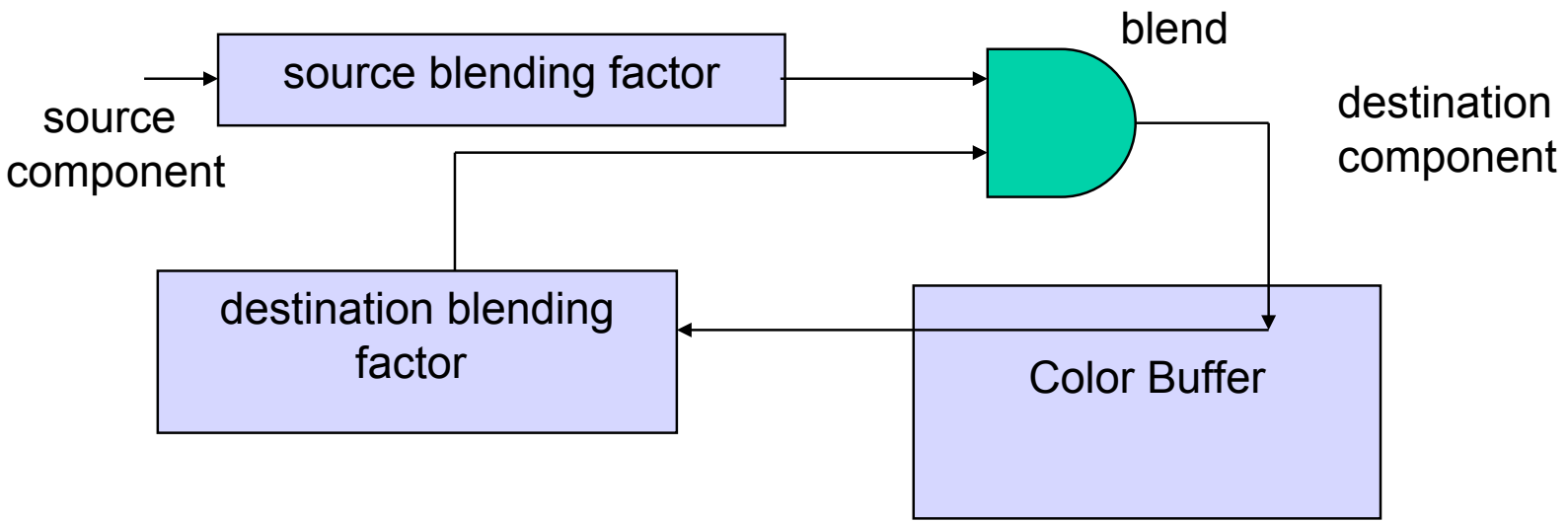
Physical Models

- Dealing with translucency in a physically correct manner is difficult due to
  - the complexity of the internal interactions of light and matter
  - Using a pipeline renderer



Writing Model

- Use A component of RGBA (or RGBa) color to store opacity
- During rendering we can expand our writing model to use RGBA values



Blending Equation

- We can define source and destination blending factors for each RGBA component

$$\mathbf{s} = [s_r, s_g, s_b, s_a]$$

$$\mathbf{d} = [d_r, d_g, d_b, d_a]$$

Suppose that the source and destination colors are

$$\mathbf{b} = [b_r, b_g, b_b, b_a]$$

$$\mathbf{c} = [c_r, c_g, c_b, c_a]$$

Blend as

$$\mathbf{c}' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_a s_a + c_a d_a]$$

OpenGL Blending and Compositing

- Must enable blending and pick source and destination factors

```
glEnable(GL_BLEND)
glBlendFunc(source_factor,
  destination_factor)
```

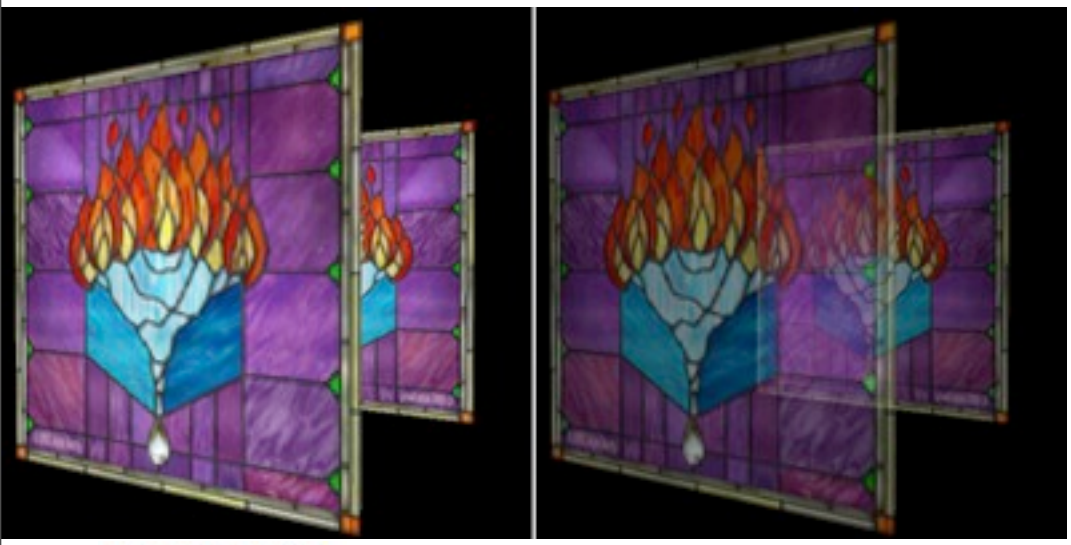
- Only certain factors supported
  - **GL_ZERO, GL_ONE**
  - **GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA**
  - **GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA**
  - See Redbook for complete list

Example

- Suppose that we start with the opaque background color $(R_0,G_0,B_0,1)$
  - This color becomes the initial destination color
- We now want to blend in a translucent polygon with color (R_1,G_1,B_1,a_1)
- Select **`GL_SRC_ALPHA`** and **`GL_ONE_MINUS_SRC_ALPHA`** as the source and destination blending factors

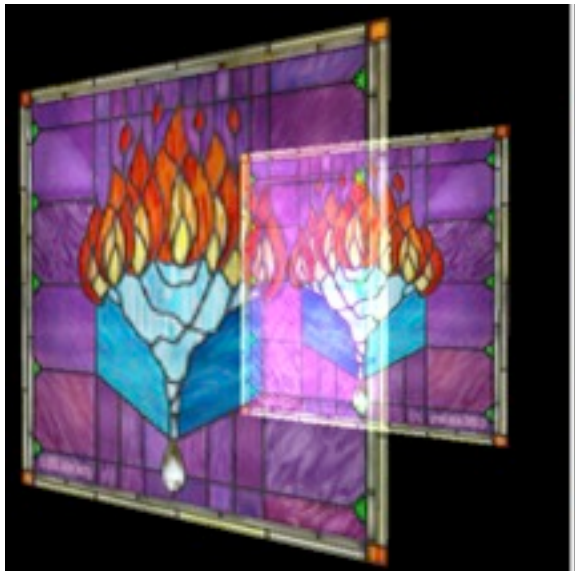
$$R'_1 = a_1 R_1 + (1 - a_1) R_0, \ldots\ldots$$

- Note this formula is correct if polygon is either opaque or transparent

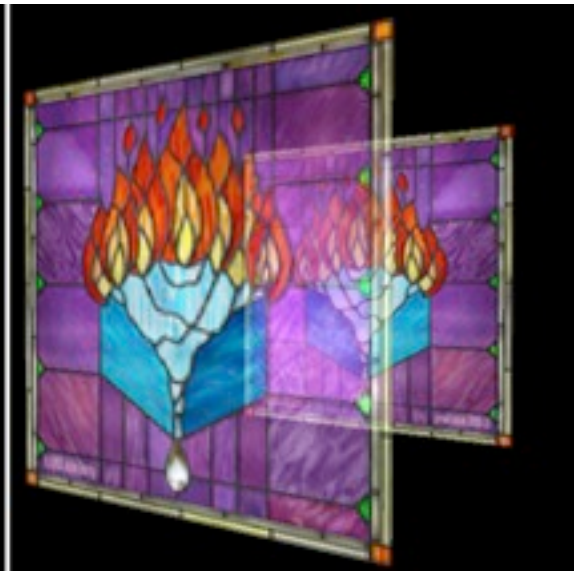


No Blending

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)



glBlendFunc(GL_ONE, GL_ONE)

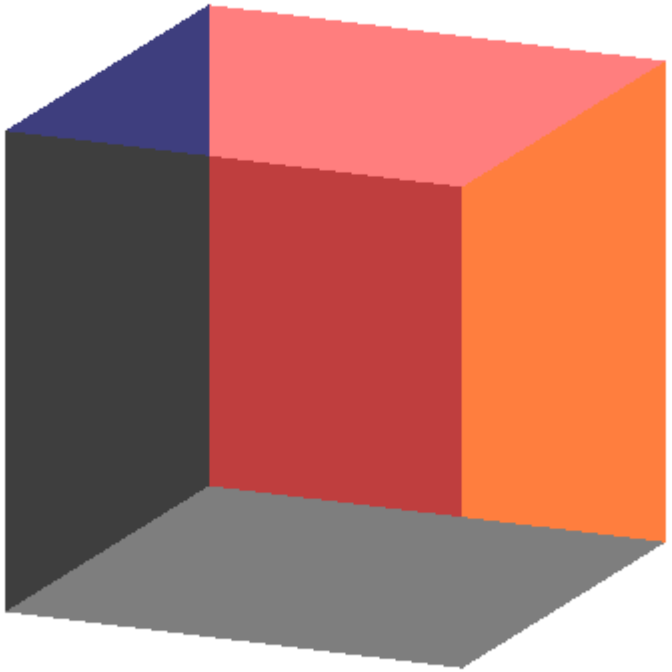


glBlendFunc(GL_ONE, GL_SRC_ALPHA)

Clamping and Accuracy

- All the components (RGBA) are clamped and stay in the range (0,1)
- However, in a typical system, RGBA values are only stored to 8 bits
  - Can easily loose accuracy if we add many components together
  - Example: add together n images
    - Divide all color components by n to avoid clamping
    - Blend with source factor = 1, destination factor = 1
    - But division by n loses bits

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent



Opague & Translucent Polygons

- Suppose that we have a group of polygons some of which are opaque and some translucent
- How do we use hidden-surface removal?
- Opaque polygons block all polygons behind them and affect the depth buffer
- Translucent polygons should not affect depth buffer
  - Render with **`glDepthMask(GL_FALSE)`** which makes depth buffer read-only
- Sort polygons first to remove order dependency

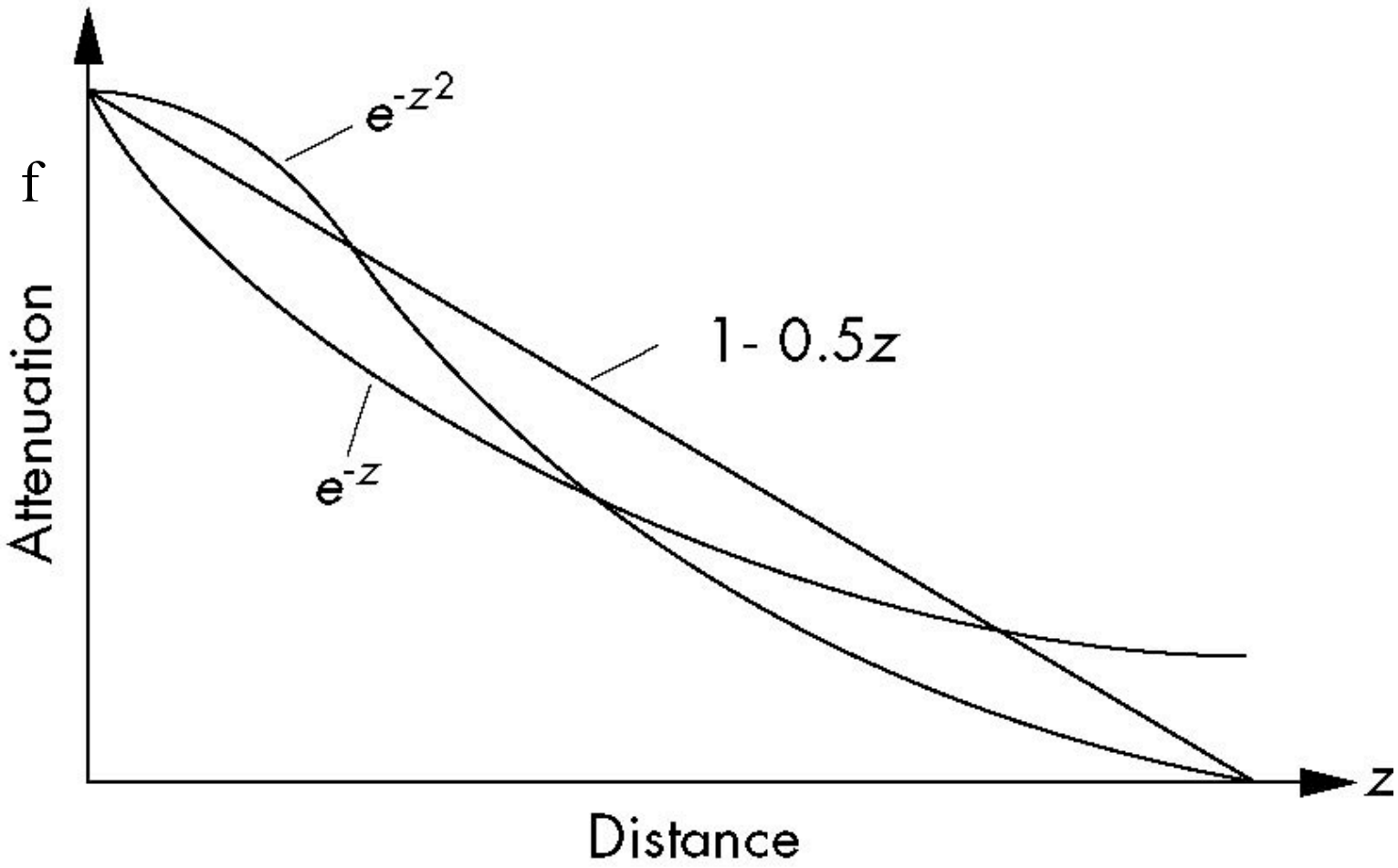
Fog

- We can composite with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
- Blend source color C_s and fog color C_f by

$$C_s' = f\, C_s + (1-f)\, C_f$$

- f is the *fog factor*
  - Exponential
  - Gaussian
  - Linear (depth cueing)

Fog Functions

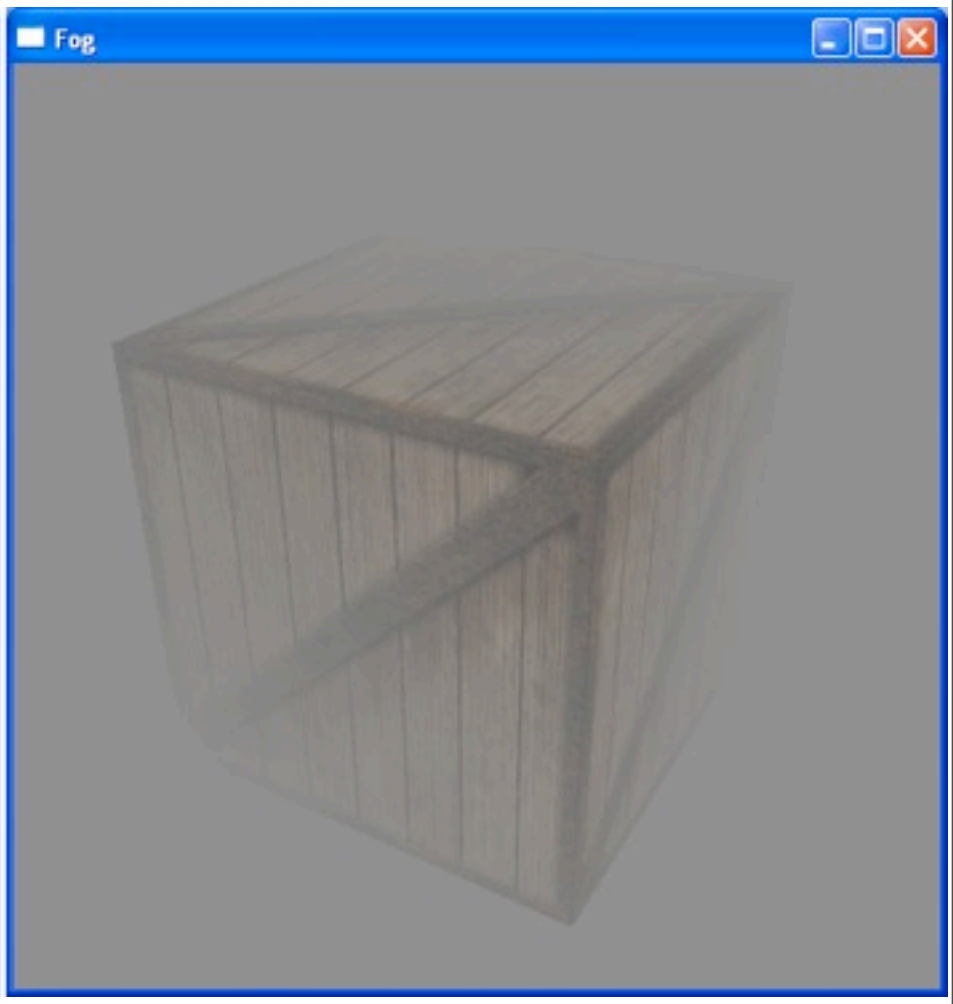


OpenGL Fog Functions

```
GLfloat fcolor[4] = {……};

glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFOgv(GL_FOG, fcolor);
```


Fog


Fog