Compositing and Blending

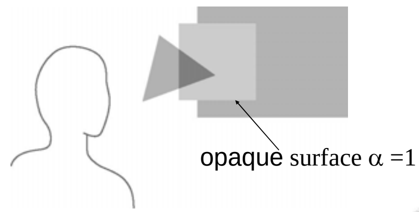
Objectives

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

Opacity and Transparency

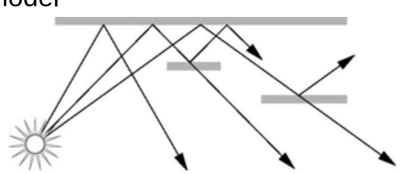
- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

  translucency = 1 – opacity (α)



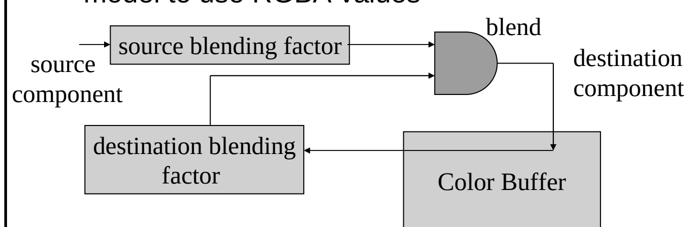
Physical Models

- 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
  - Revert to writing model



Writing Model

- Use A component of RGBA (or RGBα) 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 component
- source and destination colors

  s = $[s_r, s_g, s_b, s_\alpha]$

  d = $[d_r, d_g, d_b, d_\alpha]$
- Source and destination blending factors

  b = $[b_r, b_g, b_b, b_\alpha]$

  c = $[c_r, c_g, c_b, c_\alpha]$

Blend as

$$d' = [b_r s_r + c_r d_r,\ b_g s_g + c_g d_g,\ b_b s_b + c_b d_b,\ b_\alpha s_\alpha + c_\alpha d_\alpha]$$

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,α_1)
- Select `GL_SRC_ALPHA` and `GL_ONE_MINUS_SRC_ALPHA` as the source and destination blending factors

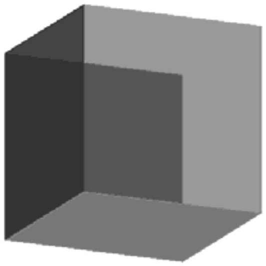
  $R'_1 = \alpha_1 R_1 + (1- \alpha_1) R_0, \ldots\ldots$
- Note this formula is correct if polygon is either opaque or transparent

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

Order Dependency

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



Opaque and 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

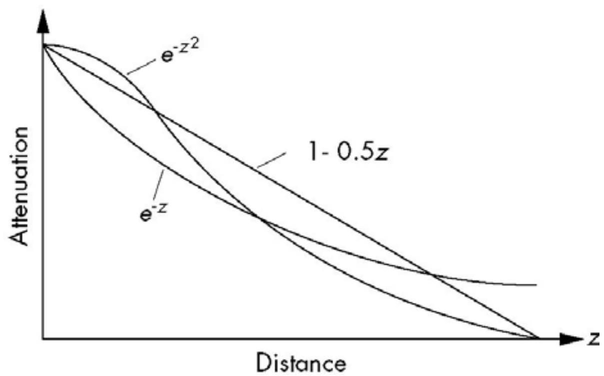
Fog

- 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

Fog density function

```
f = exp(-0.5 α²)
```

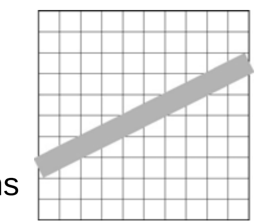
is setup in OpenGL with

```
GLfloat fcolor[4] = {……}:

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

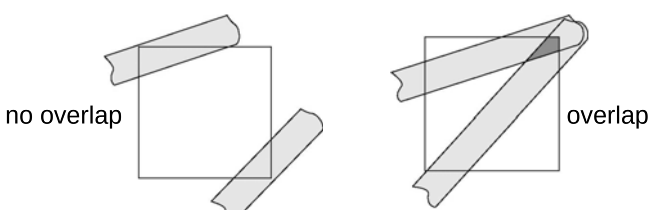

Line Aliasing

- Ideal raster line is one pixel wide
- All line segments, other than vertical and horizontal segments, partially cover pixels
- Simple algorithms color only whole pixels
- Lead to the “jaggies” or aliasing
- Similar issue for polygons



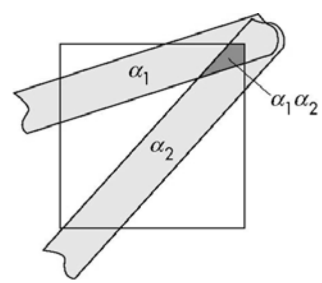
Antialiasing

- Can try to color a pixel by adding a fraction of its color to the frame buffer
  - Fraction depends on percentage of pixel covered by fragment
  - Fraction depends on whether there is overlap



Probabilistic View - Area Averaging

- Use average area $\alpha_1+\alpha_2-\alpha_1\alpha_2$ as blending factor



OpenGL Antialiasing

- Can enable separately for points, lines, or polygons

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POLYGON_SMOOTH);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```


Accumulation Buffer

- Compositing and blending are limited by resolution of the frame buffer
  - Typically 8 bits per color component
- The *accumulation buffer* is a high resolution buffer (16 or more bits per component) that avoids this problem
- Write into it or read from it with a scale factor
- Slower than direct compositing into the frame buffer

Applications

- Compositing
- Image Filtering (convolution)
- Whole scene antialiasing
- Motion effects