**Blending**

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

\[
\text{translucency} = 1 - \text{opacity (}\alpha\text{)}
\]

**Physically Correct Translucency**

Dealing with translucency in a physically correct manner is difficult due to:
- The complexity of the internal interactions of light and matter
- Limitations of fixed-pipeline rendering w/ State Machine

**Window Transparency**

- Look out a window

**Window Transparency**

- Look out a window
- What’s wrong with that?
Window Transparency

• Look out a window

• What’s wrong with that?

Screen Door Transparency

• glEnable(GL_POLYGON_STIPPLE)

Example

• Example 1
• Example 2

Compositing

• Back to Front
  \[ C_{out} = (1 - \alpha_c) C_{in} + \alpha_c C_c \]

• Front to Back
  \[ C_{out} = C_{in} + C_c \alpha_c (1 - \alpha_{in}) \]
  \[ \alpha_{out} = \alpha_{in} + \alpha_c (1 - \alpha_{in}) \]

Blending

• Combine fragments with pixel values that are already in the framebuffer

\[ \tilde{C}_r = \text{src} \tilde{C}_f + \text{dst} \tilde{C}_p \]
Blending

• Blending operation
  – Source: \( s = \{s_s, s_g, s_b, s_a\} \)
  – Destination: \( d = \{d_r, d_g, d_b, d_a\} \)
  – \( b = \{b_s, b_g, b_b, b_a\} \) source blending factors
  – \( c = \{c_r, c_g, c_b, c_a\} \) destination blending factors
  – \( d' = \{b_s s + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_a s_a + c_a d_a\} \)

OpenGL Blending and Compositing

• Must enable blending and pick source and destination factors
  \( \text{glEnable(GL_BLEND)} \)
  \( \text{glBlendFunc(srcFactor, destFactor)} \)
  \( \text{glBlendFuncSeparate(srcRGB, destRGB, srcAlpha, destAlpha)} \)

Blending Errors

• Operations are not commutative (order!)
• Operations are not idempotent
• Limited dynamic range
• Interaction with hidden-surface removal
  – Polygon behind opaque one should be hidden
  – Translucent in front of others should be composited
  – Show Demo of the problem
  – Solution?

| glBlendEquation(…), glBlendEquationSeparate(…) |
| GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL.Reverse_SUBTRACT, GL_MIN, GL_MAX |

Blending Errors

<table>
<thead>
<tr>
<th>Table 1.2 Source and Destination Blending Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
<tr>
<td>GL_ZERO</td>
</tr>
<tr>
<td>GL_SRC_COLOR</td>
</tr>
<tr>
<td>GL_SRC_ALPHA</td>
</tr>
<tr>
<td>GL_SRC1_COLOR</td>
</tr>
<tr>
<td>GL_SRC1_ALPHA</td>
</tr>
<tr>
<td>GL_CONSTANT_COLOR</td>
</tr>
<tr>
<td>GL_CONSTANT_ALPHA</td>
</tr>
<tr>
<td>GL_SRC2_COLOR</td>
</tr>
<tr>
<td>GL_SRC2_ALPHA</td>
</tr>
<tr>
<td>GL_CONSTANT_INDEX</td>
</tr>
<tr>
<td>GL_CONSTANT_INDEX</td>
</tr>
<tr>
<td>GL_BLEND_EQUATION</td>
</tr>
</tbody>
</table>

Blending Errors

• Operations are not commutative (order!)
• Operations are not idempotent
• Limited dynamic range
• Interaction with hidden-surface removal
  – Polygon behind opaque one should be hidden
  – Translucent in front of others should be composited
  – Show Demo of the problem
  – Solution?
Blending Errors

• Interaction with hidden-surface removal
  – Draw Opaque geom first, then semi-transparent
  – Use Alpha test:
    - glAlphaFunc (GL_GREATER, 0.1)
    - glEnable (GL_ALPHA_TEST)

Blending Errors

• Interaction with hidden-surface removal
  – Disable Z-test?
  – 2 polys: red (front) and blue (behind) on green background, 50% transparency
    1. Render background
    2. Render red poly
    3. Render blue poly
    What happens (z-test enabled)?

Blending Errors

• Interaction with hidden-surface removal
  – Disable Z-test?
  – 2 polys: red (front) and blue (behind) on green background, 50% transparency
    1. Render background
    2. Render red poly
    3. Render blue poly
    What happens (z-test disabled)?

Blending Errors

• Interaction with hidden-surface removal
  – Disable Z-test?
  – 2 polys: red (front) and blue (behind) on green background, 50% transparency
    1. Render background
    2. Render red poly
    3. Render blue poly
    What happens (z-test disabled)?

Blending Errors

• Interaction with hidden-surface removal
  – Polygon behind opaque one should be hidden
  – Translucent in front of others should be composited
  – Solution?
    • Two passes using alpha testing (glAlphaFunc): 1st pass
      - alpha=1 accepted, and 2nd pass alpha<1 accepted
    • make z-buffer read-only for translucent polygons (alpha<1) with glDepthMask(GL_FALSE):
    • Demo
• General Solution?
  – Just sort polygons
  • Which Space?

Sorting

• General Solution?
  – Just sort polygons
  • Which Space?
  – What About?
  – Depth Peeling

Image Dissolve?

• How to do it?

Antialiasing

• Removing the Jaggies
  \texttt{glEnable(mode)}
  - \texttt{GL_POINT_SMOOTH}
  - \texttt{GL_LINE_SMOOTH}
  - \texttt{GL_POLYGON_SMOOTH}
  – alpha value computed by computing sub-pixel coverage
  – available in both RGBA and colormap modes

• General Solution?
  – Just sort polygons
  • Which Space?
  – What About?
  – Depth Peeling (Next Time: Read the papers)
Antialiasing Revisited

- Single-polygon case first
- Set $\alpha$ value of each pixel to covered fraction
- Use destination factor of "1 – $\alpha$"
- Use source factor of "$\alpha"
- This will blend background with foreground
- Overlaps can lead to blending errors

Antialiasing with Multiple Polygons

- Initially, background color $C_0$, $a_0 = 0$
- Render first polygon; color $C_1$ fraction $\alpha_1$
  - $C_d = (1 - \alpha_1) C_0 + \alpha_1 C_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction $\alpha_2$
- If no overlap (case a), then
  - $C_d' = (1 - \alpha_2) C_d + \alpha_2 C_2$
  - $\alpha_d' = \alpha_1 + \alpha_2$

Antialiasing with Multiple Polygons

- Now assume overlap (case b)
- Average overlap is $a_1 a_2$
- So $a_d = a_1 + a_2 - a_1 a_2$
- Make front/back decision for color as usual

Antialiasing in OpenGL

- Avoid explicit $\alpha$-calculation in program
- Enable both smoothing and blending
  
  ```
  glEnable(GL_POINT_SMOOTH);
  glEnable(GL_LINE_SMOOTH);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  ```
- Can also hint about quality vs performance using `glHint(...)`

Depth Cueing and Fog

- Another application of blending
- Use distance-dependent (2) blending
  - Linear dependence: depth cueing effect
  - Exponential dependence: fog effect
- This is not a physically-based model
Example: Fog

- Fog in RGBA mode:
  \[ C = fC_0 + (1-f)C_1 \]
  
  \( f \) : depth-dependent fog factor

```c
const float fcolor[4] = {...};
glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_LINEAR);
glFogf(GL_FOG_DENSITY, 0.6);
glFogfv(GL_FOG_COLOR, fcolor);
```

[Example: Fog Tutor]

Fog Tutor

Depth Cue via Fog

Example: Depth Cue

```c
const float fcolor[4] = {0.00, 0.03, 0.02, 1.00};
glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_LINEAR);
glShadeModel(GL_FLAT, GL_SMOOTH); // per pixel, /
glFogf(GL_FOG_DENSITY, 3.0f);
glFogfv(GL_FOG_COLOR, fcolor);
glClearColor(0.0f, 0.0f, 1.0f);
```