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
  translucency = 1 – opacity (α)

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

Frame Buffer (assuming 32-bits)
  – Simple color model: R, G, B; 8 bits each
  – α-channel A, another 8 bits

• Alpha determines opacity, pixel-by-pixel
  – α = 1: opaque
  – α = 0: transparent
  – 0 < α < 1: translucent

• Blend translucent objects during rendering
• Achieve other effects (e.g., shadows)

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
  \[ \text{glBlendFunc( src, dst) } \]
  \[ \bar{C}_v = src \bar{C}_f + dst \bar{C}_p \]
Blending

- Blending operation
  - Source: \( s = [s_r, s_g, s_b, s_a] \)
  - Destination: \( d = [d_r, d_g, d_b, d_a] \)

- \( b = [b_r, b_g, b_b, b_a] \) source blending factors
- \( c = [c_r, c_g, c_b, c_a] \) destination blending factors
- \( d' = [brsr + crdr, bgsg + cgdg, bbsb + cbdb, basa + cada] \)

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`
  - `GL_ONE_MINUS_CONSTANT_SRC_ALPHA, GL_ONE_MINUS_CONSTANT_DST_ALPHA`
  - See Red Book for complete list

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


glBlendEquation(…)

- `GL_FUNC_ADD`
- `GL_FUNC_SUBTRACT`
- `GL_REVERSE_SUBTRACT`
- `GL_MIN`
- `GL_MAX`

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 blue poly
    3. Render red 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 blue poly
    3. Render red 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

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);

Sorting

- General Solution?
  - Just sort polygons
  - Which Space?
Sorting Order Matters

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 $\alpha_1$
  - $C_d = (1 – \alpha_1)C_0 + \alpha_1C_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction $\alpha_2$
  - If no overlap (case a), then
    - $C'_d = (1 – \alpha_2)C_d + \alpha_2C_2$
    - $\alpha'_d = \alpha_1 + \alpha_2$
- Now assume overlap (case b)
- Average overlap is $a_1a_2$
- So $a_2 = a_1 + a_2 – a_1a_2$
- Make front/back decision for color as usual

Antialiasing
- Removing the Jaggies
  - `glEnable(mode)`
    - `GL_POINT_SMOOTH`
    - `GL_LINE_SMOOTH`
    - `GL_POLYGON_SMOOTH`
      - alpha value computed by computing sub-pixel coverage
      - available in both RGBA and colormap modes
Antialiasing in OpenGL

- Avoid explicit $\alpha$-calculation in program
- Enable both smoothing and blending

```c
glEnable(GL_POINT_SMOOTH);
oglEnable(GL_LINE_SMOOTH);
oglEnable(GL_BLEND);
oglBlendFunc(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 (z) 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
GLfloat fcolor[4] = {...};
glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFogfv(GL_FOG_COLOR, fcolor);
```

---

Depth Cue via Fog
Example: Depth Cue

```c
float fogColor[] = [0.56, 0.52, 0.52, 1.0f];
g1.gIMultColor([0.56, 0.52, 0.52, 1.0f]);
g1.gIPosRGB(0, GL_POS_MODE, GL_POS_COLOR); /* per pixel */
g1.gIPosRGB(0, GL_POS_COLOR, fogColor);
g1.gIPosRGB(0, GL_POS_END, 1.0f);
g1.gIPosRGB(0, GL_POS_COLOR, fogColor);
g1.gIClearColor([0.56, 0.52, 0.52, 1.0f]);
```