Blending

Blending

Learn to use the A component in RGBA color for

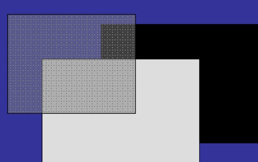
- - Blending for translucent surfaces
- - Compositing images
- - Antialiasing

Opacity and Transparency

Opaque surfaces permit no light to pass through

- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

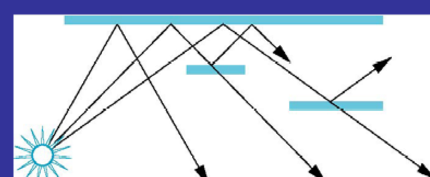
translucency = 1 – opacity (α)



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?

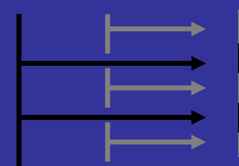
Window Transparency

- Look out a window

- What's wrong with that?

Screen Door Transparency

- glEnableGL_POLYGON_STIPPLE(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

SIGGRAPH2004

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

```
glBlendFunc( src, dst )
```

$$\bar{C}_r = src\,\bar{C}_f + dst\,\bar{C}_p$$

OpenGL

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

OpenGL Blending and Compositing

- Must enable blending and pick source and destination factors

glEnable(GL_BLEND)
glBlendFunc(srcFactor, destFactor)
glBlendFuncSeparate(
srcRGB, destRGB, srcAlpha, destAlpha)

Blending

Table 4.3 Source and Destination Blending Factors

Constant	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	$(0, 0, 0)$	0
GL_ONE	$(1, 1, 1)$	1
GL_SRC_COLOR	(R_s, G_s, B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, G_s, B_s)$	$1 - A_s$
GL_DST_COLOR	(R_d, G_d, B_d)	A_d
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1) - (R_d, G_d, B_d)$	$1 - A_d$
GL_SRC_ALPHA	(A_s, A_s, A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1) - (A_s, A_s, A_s)$	$1 - A_s$
GL_DST_ALPHA	(A_d, A_d, A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1) - (A_d, A_d, A_d)$	$1 - A_d$
GL_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, G_c, B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	$(f, f, f), f = \min(A_s, 1 - A_d)$	1
GL_SRC1_COLOR	(R_{s1}, G_{s1}, B_{s1})	A_{s1}
GL_ONE_MINUS_SRC1_COLOR	$(1, 1, 1) - (R_{s1}, G_{s1}, B_{s1})$	$1 - A_{s1}$
GL_SRC1_ALPHA	(A_{s1}, A_{s1}, A_{s1})	A_{s1}
GL_ONE_MINUS_SRC1_ALPHA	$(1, 1, 1) - (A_{s1}, A_{s1}, A_{s1})$	$1 - A_{s1}$

Blending

Table 4.3 Source and Destination Blending Factors

Constant	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	$(0, 0, 0)$	0
GL_ONE	$(1, 1, 1)$	1
GL_SRC_COLOR	(R_s, G_s, B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, G_s, B_s)$	$1 - A_s$
GL_DST_COLOR	(R_d, G_d, B_d)	A_d

void **glBlendColor**(GLclampf *red*, GLclampf *green*, GLclampf *blue*, GLclampf *alpha*);

Sets the current *red*, *blue*, *green*, and *alpha* values for use as the constant color (R_c, G_c, B_c, A_c) in blending operations.

Constant	RGB Blend Factor	Alpha Blend Factor
GL_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, G_c, B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	$(f, f, f), f = \min(A_s, 1 - A_d)$	1
GL_SRC1_COLOR	(R_{s1}, G_{s1}, B_{s1})	A_{s1}
GL_ONE_MINUS_SRC1_COLOR	$(1, 1, 1) - (R_{s1}, G_{s1}, B_{s1})$	$1 - A_{s1}$
GL_SRC1_ALPHA	(A_{s1}, A_{s1}, A_{s1})	A_{s1}
GL_ONE_MINUS_SRC1_ALPHA	$(1, 1, 1) - (A_{s1}, A_{s1}, A_{s1})$	$1 - A_{s1}$

glBlendEquation(…) glBlendEqationSeparate(…)

GL_FUNC_ADD
GL_FUNC_SUBTRACT
GL_REVERSE_SUBTRACT
GL_MIN
GL_MAX

In Table 4.4, C_s and C_d represent the source and destination colors. The S and D parameters in the table represent the source- and destination-blending factors as specified with **glBlendFunc()** or **glBlendFuncSeparate()**.

Table 4.4 Blending Equation Mathematical Operations

Blending Mode Parameter	Mathematical Operation
GL_FUNC_ADD	$C_s S + C_d D$
GL_FUNC_SUBTRACT	$C_s S - C_d D$
GL_FUNC_REVERSE_SUBTRACT	$C_d D - C_s S$
GL_MIN	$\min(C_s S, C_d D)$
GL_MAX	$\max(C_s S, C_d D)$

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 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

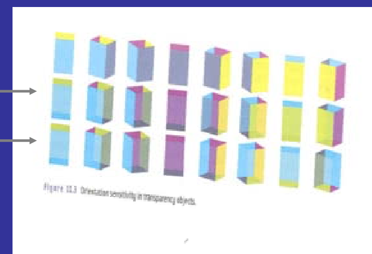
Sorting

- General Solution?
  - Just sort polygons
    - Which Space?

Sorting

Correct

Magenta
Yellow
Gray
Cyan



Sorting

- General Solution?
  - Just sort polygons
    - Which Space?
  - What About?

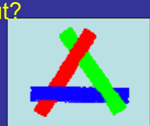
  - Depth Peeling

Image Dissolve?

- How to do it?

Sorting

- General Solution?
  - Just sort polygons
    - Which Space?
  - What About?



  - Depth Peeling (Next Time: Read the papers)

Antialiasing

SIGGRAPH2004

- 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

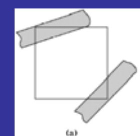
OpenGL

Antialiasing Revisited

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

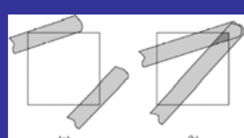
Antialiasing with Multiple Polygons

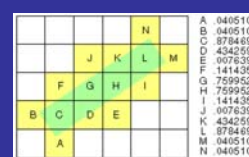
- Initially, background color $\mathbf{C}_0$, $a_0 = 0$
- Render first polygon; color C_1 fraction α_1
  - $\mathbf{C}_d = (1 - \alpha_1)\mathbf{C}_0 + \alpha_1\mathbf{C}_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction α_2
- If no overlap (case a), then
  - $\mathbf{C}'_d = (1 - \alpha_2)\mathbf{C}_d + \alpha_2\mathbf{C}_2$
  - $\alpha'_d = \alpha_1 + \alpha_2$



Antialiasing with Multiple Polygons

- Now assume overlap (case b)
- Average overlap is a_1a_2
- So $a_d = a_1 + a_2 - a_1a_2$
- Make front/back decision for color as usual





Antialiasing in OpenGL

- Avoid explicit α-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 (z) blending
  - Linear dependence: depth cueing effect
  - Exponential dependence: fog effect
  - This is not a physically-based model

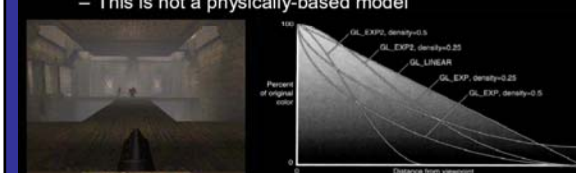


Figure 6-4 Fog-Density Equations

Example: Fog

- Fog in RGBA mode:

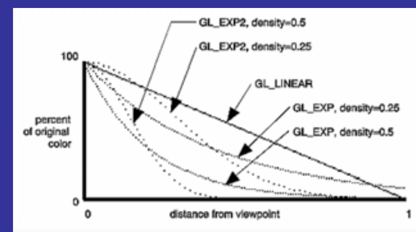
  $C = fC_i + (1\text{-}f)C_f$

  – f : depth-dependent fog factor

```
GLfloat fcolor[4] = {...};
glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFogfv(GL_FOG_COLOR, fcolor);
```

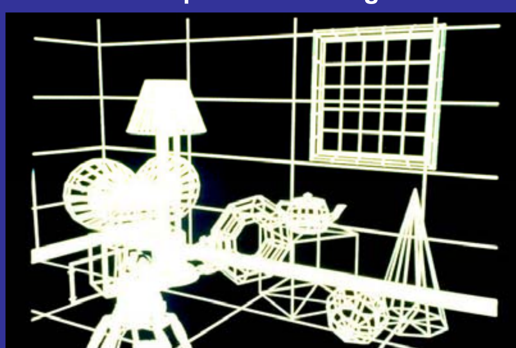
[Example: Fog Tutor]

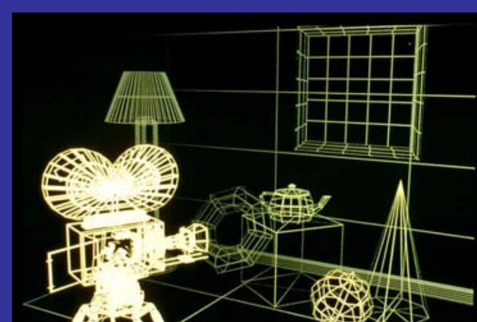
Fog



Fog Tutor

Depth Cue via Fog





Example: Depth Cue

Example

```
float fogColor[] = {0.0f, 0.0f, 0.0f, 1.0f};
gl.glEnable(GL_FOG);
gl.glFogi (GL_FOG_MODE, GL_LINEAR);
gl.glHint (GL_FOG_HINT, GL_NICEST);  /*  per pixel   */
gl.glFogf (GL_FOG_START, 3.0f);
gl.glFogf (GL_FOG_END, 5.0f);
gl.glFogfv (GL_FOG_COLOR, fogColor);
gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```