CIS 441/541: Introduction to Computer Graphics
Lecture 15: transparency

May 24th, 2013
Hank Childs, University of Oregon
- OH today 2-4
- Normal schedule next week
  - Tues 10-12
  - Weds 2-4
Final projects

- **Ideas:**
  - Video games
  - Visualizations
  - Modeling
  - Screen savers

- **Final project proposal:**
  - One to two paragraphs
  - Not graded
  - Please send in ASAP

- **Other**
  - I will ask how many hours you have spent on this project
  - Should be starting on this project ASAP
The rest of this quarter

- May 17th / Lecture 13: geometry creation
- May 22nd / Lecture 14: buffers in GL
- May 24th / Lecture 15: transparency
- May 29th / Lecture 16: terrain rendering
- May 31st / Lecture 17: parallel rendering, quaternions???
- June 5th: Exam (worth 25% of your grade)
  - What will be on the Exam?
- June 7th / Lecture 18: Research with GPUs
What will be on the Final?

- Closed book
- I do expect you know how to shade things
- I do expect that you can write reasonable facsimiles of OpenGL programs that would come close to compiling
- I don’t expect you to have the view transformation matrix or rotation memorized
  - Scale and translate: maybe
Some of my favorite 2B's so far...
Some of my favorite 2B’s so far...
Some of my favorite 2B’s so far...
If you don’t like the game, change the rules…

```c
# glDisable(GL_LIGHTING);
# glBegin(GL_LINES);
# glColor3ub(255, 0, 0);
# glVertex3f(25, 0, 0);
# glColor3ub(128, 0, 0);
# glVertex3f(-25, 0, 0);
# glColor3ub(0, 255, 0);
# glVertex3f(0, 25, 0);
# glColor3ub(0, 128, 0);
# glVertex3f(0, -25, 0);
# glColor3ub(0, 0, 255);
# glVertex3f(0, 0, 25);
# glColor3ub(0, 0, 128);
# glVertex3f(0, 0, -25);
# glEnd();
# glEnable(GL_LIGHTING);
```
Jobs

- Internships are very important!
- Interest in Portland jobs?
What is multi-pass rendering?

- Multipass rendering refers to a set of techniques in 3D computer graphics.
- In multipass rendering, a single 3D object (or scene) is rendered multiple times.
  - Each time the object is drawn, an additional aspect of object's appearance is calculated and combined with the previous results.
  - The process of rendering the object is called a pass.
Multi-pass rendering discussion for…

- Mirrors
- Shadow maps
  - Possible to do with stencils, but gets very involved
- What are environment maps?
- What are skyboxes?
- Also discuss FBOs
Shadow Maps

1. Render scene from light source view. Store depth value in depth map.

2. Render scene from camera view. Compare depth til pixel with value in map.
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 ($\alpha$)

Opaque surface $\alpha = 1$
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 RGB\(\alpha\)) color to store opacity
- During rendering we can expand our writing model to use RGBA values

source component → source blending factor → blend → destination component

destination blending factor

Color Buffer
Blending Equation

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

\[ s = [s_r, s_g, s_b, s_\alpha] \]
\[ d = [d_r, d_g, d_b, d_\alpha] \]

Suppose that the source and destination colors are

\[ b = [b_r, b_g, b_b, b_\alpha] \]
\[ c = [c_r, c_g, c_b, c_\alpha] \]

Blend as

\[ c' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha] \]
OpenGL Blending and Compositing

• Must enable blending and pick source and destination factors

\[
\text{glEnable(GL\_BLEND)}
\]

\[
\text{glBlendFunc(source\_factor, destination\_factor)}
\]

• Only certain factors supported

- \text{GL\_ZERO, GL\_ONE}
- \text{GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA}
- \text{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 \((R_1, G_1, B_1, \alpha_1)\)
• Select \texttt{GL\_SRC\_ALPHA} and \texttt{GL\_ONE\_MINUS\_SRC\_ALPHA} as the source and destination blending factors
  \[ R'_1 = \alpha_1 R_1 + (1 - \alpha_1) R_0, \ldots \]
• Note this formula is correct if polygon is either opaque or transparent
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
Code exploration
glBlendFunc — specify pixel arithmetic

C Specification

```c
void glBlendFunc(GLenum sfactor,
                 GLenum dfactor);

void glBlendFunci(GLuint buf,
                 GLenum sfactor,
                 GLenum dfactor);
```

Parameters

`buf`

For `glBlendFunci`, specifies the index of the draw buffer for which to set the blend function.

`sfactor`

Specifies how the red, green, blue, and alpha source blending factors are computed. The initial value is `GL_ONE`.

`dfactor`

Specifies how the red, green, blue, and alpha destination blending factors are computed. The following symbolic constants are accepted: `GL_ZERO, GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA, GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, and GL_ONE_MINUS_CONSTANT_ALPHA`. The initial value is `GL_ZERO`. 
In the table and in subsequent equations, source and destination color components are referred to as \((R_s, G_s, B_s, A_s)\) and \((R_d, G_d, B_d, A_d)\). The color specified by \texttt{glBlendColor} is referred to as \((R_c, G_c, B_c, A_c)\). They are understood to have integer values between 0 and \((kR, kG, kB, kA)\), where

\[ k_c = (2^{mc}) - 1 \]

and \((mR, mG, mB, mA)\) is the number of red, green, blue, and alpha bitplanes.

Source and destination scale factors are referred to as \((sR, sG, sB, sA)\) and \((dR, dG, dB, dA)\). The scale factors described in the table, denoted \((fR, fG, fB, fA)\), represent either source or destination factors. All scale factors have range \([0, 1]\).

Parameter

\[
\begin{align*}
\text{GL_ZERO} & (0, 0, 0, 0) \\
\text{GL_ONE} & (1, 1, 1, 1) \\
\text{GL_SRC_COLOR} & (Rs / kR, Gs / kG, Bs / kB, As / kA) \\
\text{GL_ONE_MINUS_SRC_COLOR} & (1, 1, 1, 1) - (Rs / kR, Gs / kG, Bs / kB, As / kA) \\
\text{GL_DST_COLOR} & (Rd / kR, Gd / kG, Bd / kB, Ad / kA) \\
\text{GL_ONE_MINUS_DST_COLOR} & (1, 1, 1, 1) - (Rd / kR, Gd / kG, Bd / kB, Ad / kA) \\
\text{GL_SRC_ALPHA} & (As / kA, As / kA, As / kA, As / kA) \\
\text{GL_ONE_MINUS_SRC_ALPHA} & (1, 1, 1, 1) - (As / kA, As / kA, As / kA, As / kA) \\
\text{GL_DST_ALPHA} & (Ad / kA, Ad / kA, Ad / kA, Ad / kA) \\
\text{GL_ONE_MINUS_DST_ALPHA} & (1, 1, 1, 1) - (Ad / kA, Ad / kA, Ad / kA, Ad / kA) \\
\text{GL_SRC_ALPHA_SATURATE} & (i, i, i, i) \\
\text{GL_CONSTANT_COLOR} & (Rc, Gc, Bc, Ac) \\
\text{GL_ONE_MINUS_CONSTANT_COLOR} & (1, 1, 1, 1) - (Rc, Gc, Bc, Ac) \\
\text{GL_CONSTANT_ALPHA} & (Ac, Ac, Ac, Ac) \\
\text{GL_ONE_MINUS_CONSTANT_ALPHA} & (1, 1, 1, 1) - (Ac, Ac, Ac, Ac)
\end{align*}
\]
How do you sort?

- 1) Calculate depth of each triangle center.
- 2) Sort based on depth
  - Not perfect, but good

In practice: sort along X, Y, and Z and use “dominant axis” and only do “perfect sort” when rotation stops
But there is a problem...
The rest of this quarter

- May 17th / Lecture 13: geometry creation
- May 22nd / Lecture 14: buffers in GL
- May 24th / Lecture 15: transparency
- May 29th / Lecture 16: terrain rendering
- May 31st / Lecture 17: parallel rendering, quaternions???
- June 5th: Exam (worth 25% of your grade)
- June 7th / Lecture 18: Research with GPUs