Announcements

- OH canceled today
- Hope that today’s lecture answers project questions
  - If not, focus on them next week in OH
    - … and can hold some early
HAND BACK AND DISCUSS MIDTERMS
Final projects
Animation and Acceleration
Multi-pass Rendering
Transparency
Outline

- Final projects
- Animation and Acceleration
- Multi-pass Rendering
- Transparency
Guidelines

- Do something that excites you
- Should be a significant project
- Should involve graphics in a serious way
  - Including implementation of graphics code
  - Example: Use program to generate geometry, but ALSO add graphics codes to see that geometry in a cool way
- I don’t care what technology you use: VTK, GLUT, GL, etc.
Ideas

- Screen saver
- Modeling (+ graphics)
- Video Game
- Visualization
- Advanced animation
- Advanced graphics techniques
- really anything…
Deliverable:
- Email me a written proposal
- 1-2 paragraphs
- In word/PDF, not an email
- not graded

Goal: make sure we don’t find out we disagree on the day of the final

Deadline: Nov 26
- Will turn them around as they come in
  - (so you can hand it in earlier if you want)

Office Hours will change to support your projects
Outline

- Final projects
- Animation and Acceleration
- Multi-pass Rendering
- Transparency
Keyframe Animation

- Idea: pick “key frames”
- Specify how to do those frames in entirety
- Remaining frames are interpolated between keyframes
Keyframe Animation: Example

- 1000 frames

- Frame 0:
  - ball sits on floor, at position (0,0,0)

- Frame 1000:
  - ball at position (1,1,1)

- Frame i, i=1-999:
  - ball at position (0.001*i, 0.001*i, 0.001*i)
Keyframe Animation: Example

- 1000 frames
- Frame 0:
  - ball sits on floor, at position (0,0,0)
- Frame 100:
  - ball sits on floor, at position (0,0,0)
- Frame 1100:
  - ball at position (1,1,1)
- Frame \(i\), \(i=1-99\):
  - ball at position (0,0,0)
- Frame \(i\), \(i=101-1099\):
  - ball at position \((0.001*(i-100), 0.001*(i-100), 0.001*(i-100))\)
Keyframe Animation: Example

- 1000 frames

- Frame 0:
  - ball1 sits on floor, at position (0,0,0), ball2 at position (1,1,1)

- Frame 100:
  - ball sits on floor, at position (0,0,0)

- Frame 1100:
  - ball at position (1,1,1), ball2 at position (2.1, 2.1, 2.1)

- Frame $i$, $i=1-99$:
  - ball at position (0,0,0)
  - ball2 at position ($1+0.001*i$, $1+0.001*i$, $1+0.001*i$)

- Frame $i$, $i=101-1099$:
  - ball at position ($0.001*(i-100)$, $0.001*(i-100)$, $0.001*(i-100)$)
  - ball2 at position ($1+0.001*i$, $1+0.001*i$, $1+0.001*i$)
If object is still, and then abruptly starts moving, then, it is visually disturbing

Previous example:

![Graph showing velocity over frame]
If object is still, and then abruptly starts moving, then, it is visually disturbing.

Previous example:

- Velocity begins at 0.
- Acceleration window (could be many frames or few).
- Slope is 0 at end (matches upcoming stretch of constant velocity).
- Ultimately has constant velocity.
- Have to go a little faster to make up for acceleration window.
Acceleration

- If object is still, and then abruptly starts moving, then, it is visually disturbing

- Previous example:
Desired curve

- value(F2) = V, derivative(F2) = 0
- value(F1) = 0

What curve can we choose?
Acceleration

- **Desired curve**
  - \(\text{value}(F2) = V, \text{derivative}(F2) = 0\)
  - \(\text{value}(F1) = 0\)

- **Candidate:**
  - \(\sin\left(\frac{(F-F1)}{(F2-F1)} \cdot \frac{\pi}{2}\right) \cdot V\)
  - \(F1:\)
    - \(\text{value}(F1) = \sin\left(\frac{0}{(F2-F2)} \cdot \frac{\pi}{2}\right) \cdot V = \sin(0) \cdot V = 0 \cdot V = 0\)
  - \(F2:\)
    - \(\text{value}(F2) = \sin((F2-F1)/(F2-F1) \cdot \frac{\pi}{2}) \cdot V = \sin(\frac{\pi}{2}) \cdot V = 1 \cdot V\)
    - \(\text{derivative}(F2) = \cos(\frac{\pi}{2}) \cdot V = 0\)
If object is still, and then abruptly starts moving, then, it is visually disturbing.

What is V?
If object is still, and then abruptly starts moving, then, it is visually disturbing.

Distance traveled for conventional = \((F_4-F_1)\times V'\)

What is distance traveled with acceleration?
\[
\text{integral}_{F1}^{F2} (V \cdot \sin(F)) \, dF + (F3-F2) \cdot V + \text{integral}_{F3}^{F4} (V \cdot \sin(F)) \, dF \quad \text{... and} \quad = (F4-F1) \cdot V' 
\]
In Python

```python
# This allows us to have acceleration, which makes the transition between
# effects like fly-throughs and clips not feel jumpy.
def SineParameterize(nFrames, curFrame, ramp):
    # We are going to construct a function that has sine curves at either
    # end and a flat ramp in the middle. We will then parameterize space
    # by determining what portion of the total area has been covered by
    # frame "curFrame".
    nFrames -= 1
    if 2*ramp > nFrames:
        print "Ramp too large -- correcting"
        ramp = nFrames / 2
    if ramp <= 0:
        return 1.
    if nFrames <= 0:
        return 1.
    nNonRamp = nFrames - 2*ramp
    # determine the height of our function
    height=1./((float(nNonRamp) + 4*float(ramp))/math.pi)
    if curFrame < ramp:
        factor=2*height*ramp/math.pi
        eval=math.cos((math.pi/2.)*(float(curFrame)/float(ramp)))
        return (1. - eval)*factor
    elif curFrame > nFrames-ramp:
        amount_left = nFrames-curFrame
        factor=2*height*ramp/math.pi
        eval=math.cos((math.pi/2.)*(float(amount_left)/float(ramp)))
        return 1. - (1. - eval)*factor
    else:
        amount_in_quad=curFrame-ramp
        quad_part=amount_in_quad*height
        curve_part=height*(2*ramp)/math.pi
        return quad_part+curve_part
```
NOW EXPLORE CODE
Outline

- Final projects
- Animation and Acceleration
- Multi-pass Rendering
- Transparency
A screenshot from a program which uses that stencil buffer to change the color of shapes as they pass over other shapes. The stencil buffer is filled with 1s wherever a white stripe is drawn and 0s elsewhere. Two versions of each oval, square, or triangle are then drawn, and the stencil buffer is used to decide per-pixel which object to draw. A black colored shape is drawn where the stencil buffer is 0, and a white shape is drawn where the buffer is 1.
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 the object's appearance is calculated and combined with the previous results.
  - The process of rendering the object is called a pass.
Accumulation buffer: motion blur
Multi-pass rendering discussion for...

- Mirrors
- What are environment maps?
- What are skyboxes?

source: wikipedia

source: Brad Syrie final project
Shadow Maps

1. Render scene from light-source-view. Store depth value in depth map.
2. Render scene from camera-view. Compare depth to pixel value in map.
Outline

- Final projects
- Animation and Acceleration
- Multi-pass Rendering
- Transparency
Compositing and Blending

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
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 $\alpha = 1$
Transparency

• If you have an opaque red square in front of a blue square, what color would you see?
  – Red

• If you have a 50% transparent red square in front of a blue square, what color would you see?
  – Purple

• If you have a 100% transparent red square in front of a blue square, what color would you see?
  – Blue
(One) Formula For Transparency

- Front = (Fr,Fg,Fb,Fa)
  - $a = \alpha$, transparency factor
    - Sometimes percent
    - Typically 0-255, with 255 = 100%, 0 = 0%
- Back = (Br,Bg,Bb,Ba)
- Equation = $(Fa*Fr+(1-Fa)*Br,$
  $Fa*Fg+(1-Fa)*Bg,$
  $Fa*Fb+(1-Fa)*Bb,$
  $Fa+(1-Fa)*Ba)$

**Alpha component is important!** Any observations?
Transparency

• If you have an 25% transparent red square (255,0,0) in front of a blue square (0,0,255), what color would you see (in RGB)?
  – (192,0,64)

• If you have an 25% transparent blue square (0,0,255) in front of a red square (255,0,0), what color would you see (in RGB)?
  – (64,0,192)
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
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 \((R_1, G_1, B_1, \alpha_1)\)
• Select \texttt{GL\_SRC\_ALPHA} and \texttt{GL\_ONE\_MINUS\_SRC\_ALPHA}\n
as the source and destination blending factors
\[
R'_1 = \alpha_1 R_1 + (1- \alpha_1) R_0, \quad \ldots
\]
• 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
glBlendFunc

---

**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 \((k_R, k_G, k_B, k_A)\), where

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

and \((m_R, m_G, m_B, m_A)\) is the number of red, green, blue, and alpha bitplanes.

Source and destination scale factors are referred to as \((s_R, s_G, s_B, s_A)\) and \((d_R, d_G, d_B, d_A)\). The scale factors described in the table, denoted \((f_R, f_G, f_B, f_A)\), represent either source or destination factors. All scale factors have range \([0, 1]\).

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>((f_R, f_G, f_B, f_A))</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{GL_ZERO}</td>
<td>((0, 0, 0, 0))</td>
</tr>
<tr>
<td>\texttt{GL_ONE}</td>
<td>((1, 1, 1, 1))</td>
</tr>
<tr>
<td>\texttt{GL_SRC_COLOR}</td>
<td>((R_s / k_R, G_s / k_G, B_s / k_B, A_s / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_SRC_COLOR}</td>
<td>((1, 1, 1, 1) - (R_s / k_R, G_s / k_G, B_s / k_B, A_s / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_DST_COLOR}</td>
<td>((R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_DST_COLOR}</td>
<td>((1, 1, 1, 1) - (R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_SRC_ALPHA}</td>
<td>((A_s / k_A, A_s / k_A, A_s / k_A, A_s / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_SRC_ALPHA}</td>
<td>((1, 1, 1, 1) - (A_s / k_A, A_s / k_A, A_s / k_A, A_s / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_DST_ALPHA}</td>
<td>((A_d / k_A, A_d / k_A, A_d / k_A, A_d / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_DST_ALPHA}</td>
<td>((1, 1, 1, 1) - (A_d / k_A, A_d / k_A, A_d / k_A, A_d / k_A))</td>
</tr>
<tr>
<td>\texttt{GL_SRC_ALPHA_SATURATE}</td>
<td>((i, i, i, i))</td>
</tr>
<tr>
<td>\texttt{GL_CONSTANT_COLOR}</td>
<td>((R_c, G_c, B_c, A_c))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_CONSTANT_COLOR}</td>
<td>((1, 1, 1, 1) - (R_c, G_c, B_c, A_c))</td>
</tr>
<tr>
<td>\texttt{GL_CONSTANT_ALPHA}</td>
<td>((A_c, A_c, A_c, A_c))</td>
</tr>
<tr>
<td>\texttt{GL_ONE_MINUS_CONSTANT_ALPHA}</td>
<td>((1, 1, 1, 1) - (A_c, A_c, A_c, A_c))</td>
</tr>
</tbody>
</table>
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...
Depth Peeling

- a multi-pass technique that renders transparent polygonal geometry without sorting

- Pass #1:
  - render as opaque, but note opacity of pixels placed on top
  - treat this as “top layer”
  - save Z-buffer and treat this as “max”

- Pass #2:
  - render as opaque, but ignore fragments beyond “max”

- repeat, repeat...
Happy Thanksgiving