Computer Graphics Programming I

- Agenda:
  - Fog
  - Framebuffer operations
    - Blending
    - Alpha test
  - Multi-pass rendering
  - Term projects assigned!!!

Fog

- Typical fog...as objects get farther away from the camera, they become more fog colored.
  - Eventually objects completely fade to fog color
  - Controlled by a fog weight on the range [0, 1]
  - Applied *after* texturing and *after* separate specular
    - Enabled with `GL_FOG`.
- Can be used for other related effects:
  - In dark environments, distant objects are darker
  - Underwater objects fade to the water color
    - Notice that the only difference is the *color* of the fog!

Fog Parameters

- Fog has 4 main parameters:
  - `GL_FOG_START`: Distance to fog start
  - `GL_FOG_END`: Distance to maximum fog density
    - These parameters only apply to `GL_LINEAR` fog
  - `GL_FOG_DENSITY`: Density (surprise!) of the fog
    - This parameter only applies to `GL_EXP` and `GL_EXP2` fog
  - `GL_FOG_COLOR`
- All parameters set via `glFog{if}[v]`

Fog Modes

- Fog is applied according to one of 3 equations:
  - `GL_LINEAR`: $\frac{end - c}{end - start}$
  - `GL_EXP`: $e^{(-d \times c)}$
  - `GL_EXP2`: $e^{(-d \times c)^2}$
- The mode, start, end, and density control how OpenGL calculates the fog weight from the Z value
  - Somewhat like lighting
- Set as the `GL_FOG_MODE` parameter of `glFogi`

Explicit Fog Coordinate

- Instead of allowing the GL to calculate a fog coordinate, specify one explicitly
  - `GL_EXT_fog_coord` or version 1.4
  - Set `GL_FOG_COORD_SRC` to `GL_FOG_COORD` to enable
    - Set it to `GL_FRAGMENT_DEPTH` to disable
- Fog coord specified by `glFogCoord1{fd}[v]`
  - Coordinate is the distance used in the fog equations
  - **Not** the fog weight!

Height-based Fog

- Fog factor is given by:

$$e^{-\int_A^B \alpha(t)\,dt}$$

Where:

α is the fog density function

A and B are points in space

- This integral gives the “optical depth”.
- One simplifying assumption: α depends only on height

Height-based Fog (cont.)

- Two components to the optical distance between the eye and the fogged point:
  - Change in altitude: $\Delta y = y_{point} - y_{eye}$
  - Distance in the plane: $\Delta D = \sqrt{((X_{point} - X_{eye})^2 + (Z_{point} - Z_{eye})^2)}$
- Two important cases:
  - $\Delta y = 0$: $\Delta D \times y_{point}$
  - $\Delta y \neq 0$: $\sqrt{1+\left(\frac{\Delta D}{\Delta y}\right)^2} \times \int_{y_{eye}}^{y_{point}} \alpha(y)\, dy$

Height-based Fog (cont.)

- Store a look-up where the value at an element *n* is:

$$\int_{-\infty}^{n} \alpha(y)\,dy$$

- To calculate the integral over y_{eye} to y_{point}, simply calculate `table[y_point]-table[y_eye]`
  - This kind of table is called a *summed-area table*, and it is incredibly useful!

References

http://developer.nvidia.com/object/shadows_transparency_fog.html

http://mrl.nyu.edu/~perlin/experiments/ball/

- Very cool example of what can be done with explicit fog coordinates...by one of the legends of computer graphics

http://mrl.nyu.edu/~perlin/experiments/gabor/

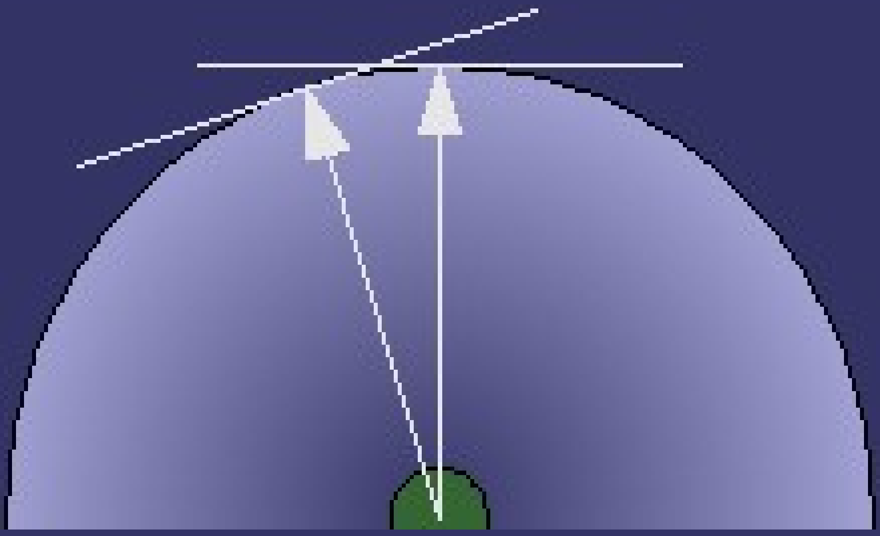
- Some of the theory behind the above Java applet.

Legakis, J. Fast multi-layer fog. In *ACM SIGGRAPH 98 Conference Abstracts and Applications* (Orlando, Florida, United States, July 19 - 24, 1998). SIGGRAPH '98. ACM, New York, NY.

- Great paper, but not available on-line. :(

Radial Fog

- `GL_FRAGMENT_DEPTH` based fog generates incorrect values away from the screen center
  - It uses the distance from the near plane instead of the distance from the eye.
  - Could fix by calculating true distance on CPU and using explicit fog coordinates.



Radial Fog (cont.)

- GL_NV_fog_distance also fixes this.
  - Adds new fog param GL_FOG_DISTANCE_MODE_NV
  - Three possible values:
    - GL_EYE_PLANE: Fog coord is Z value in eye-space
    - GL_EYE_PLANE_ABSOLUTE_NV: Fog coord is |Z| value in eye-space
      - This is the “usual” approximation allowed by the OpenGL spec
    - GL_EYE_RADIAL_NV: Fog coord is the distance of the point to the eye

Blending

- Typically used for one of a few operations:
  - Translucent / transparent objects
    - In general this is a hard problem
    - Objects must be rendered back to front
    - Polygons can't intersect
  - Antialiasing
    - Especially useful for fonts
  - 2D compositing
    - Uh...you've seen OS X, right?
  - Multi-pass rendering

Blend Function

$$C_{src} \times F_{src} + C_{dst} \times F_{dst}$$

Fragment color

Source blending factor:

- GL_ZERO
- GL_ONE
- GL_SRC_ALPHA
- GL_ONE_MINUS_SRC_ALPHA
- GL_DST_COLOR
- GL_ONE_MINUS_DST_COLOR
- GL_DST_ALPHA
- GL_ONE_MINUS_DST_ALPHA
- GL_SRC_ALPHA_SATURATE

$F_{src} = \min(A_s, 1 - A_d)$

Destination blending factor:

- GL_ZERO
- GL_ONE
- GL_SRC_COLOR
- GL_ONE_MINUS_SRC_COLOR
- GL_SRC_ALPHA
- GL_ONE_MINUS_SRC_ALPHA
- GL_DST_ALPHA
- GL_ONE_MINUS_DST_ALPHA

Color already in framebuffer

GL_EXT_blend_color

$$C_{src} \times F_{src} + C_{dst} \times F_{dst}$$

Source blending factor:

- GL_CONSTANT_COLOR_EXT
- GL_ONE_MINUS_CONSTANT_COLOR_EXT
- GL_CONSTANT_ALPHA_EXT
- GL_ONE_MINUS_CONSTANT_ALPHA_EXT

Destination blending factor:

- GL_CONSTANT_COLOR_EXT
- GL_ONE_MINUS_CONSTANT_COLOR_EXT
- GL_CONSTANT_ALPHA_EXT
- GL_ONE_MINUS_CONSTANT_ALPHA_EXT

- Constant color set with `glBlendColorEXT`.
- Included in version 1.4 and `GL_ARB_imaging`
  - These versions drop EXT from names.

GL_NV_blend_square

$$C_{src} \times F_{src} + C_{dst} \times F_{dst}$$

Source blending factor:

- GL_SRC_COLOR
- GL_ONE_MINUS_SRC_COLOR

Destination blending factor:

- GL_DST_COLOR
- GL_ONE_MINUS_DST_COLOR

➲ Also included with core version 1.4.

Blend Equation

$$C_{src} \times F_{src} + C_{dst} \times F_{dst}$$

- Several extensions allow different math:
  - GL_EXT_blend_subtract: GL_SUBTRACT, GL_REVERSE_SUBTRACT
  - GL_EXT_blend_minmax: GL_MIN, GL_MAX
    - Both included in 1.4 and GL_ARB_imaging.
- Equation set with glBlendEquation.
- Others exist, but are *very* rare.

Separate Blend Function / Equation

- Function and equation apply to RGB and A.
- `GL_EXT_blend_function_separate` allows a different function for color and alpha.
  - Adds `glBlendFuncSeparateEXT`
  - Included in core version 1.4.
- `GL_EXT_blend_equation_separate` allows a different equation for color and alpha.
  - Adds `glBlendEquationSeparateEXT`
  - Included in core version 2.0.

References

http://en.wikipedia.org/wiki/Alpha_compositing

- Good background of general alpha blending theory

http://developer.nvidia.com/object/order_independent_transparency.html

- Solves the ordering problem, but requires features we won't cover this term.
- Will be *required* reading for VGP352. :)

Alpha Test

- Yet another way to reject fragments
  - Enable with `GL_ALPHA_TEST`
  - Set test function and reference value with `glAlphaFunc`
    - Same set of functions available as with depth testing.
  - Compares fragment alpha with the reference value
    - If the test fails, the fragment is rejected.
    - Similar to depth testing
- Alpha testing occurs *before* stencil testing
  - ...and stencil testing happens before depth testing

Multi-pass Rendering

- *Please...*no 5th Element jokes.
- Multi-pass rendering is used more work has to be done than the hardware can handle.
  - Example: produce correct specular highlights on textured objects *without* `GL_EXT_separate_specular`
  - Example: want to do bump-mapped shading for diffuse and specular, but only have 2 texture units

Multi-pass Rendering (cont.)

- Divide rendering into steps that the texture combiners can do and that are separated by math that the blender can do
  - Example: Perform diffuse textured pass. Configure blender to add fragment color to framebuffer. Finally, perform specular-only pass.

Problems with Multi-pass

- Why do we want to avoid multi-passing?

Problems with Multi-pass

- Why do we want to avoid multi-passing?
  - It's slower.
    - The memory for each pixel gets accessed multiple times
    - Have to process the same geometry multiple times
    - Have to change state (e.g., textures) between passes

Problems with Multi-pass

- Why do we want to avoid multi-passing?
  - It's slower.
    - The memory for each pixel gets accessed multiple times
    - Have to process the same geometry multiple times
    - Have to change state (e.g., textures) between passes
  - Less accurate
    - Common best-case framebuffer has 8-bits of precision per color component
    - Common best-case texture combiners have 12-bits of precision per color component

Problems with Multi-pass (cont.)

- Why do we want to avoid multi-passing?
  - Can't always achieve desired result
    - Doesn't work well with translucent objects
    - Can't alway break the math down

References

http://www.bluesnews.com/cgi-bin/finger.pl?id=1&time=20000429013039

- Interesting comments by John Carmack about color precision in multi-pass rendering

Next week...

- Faster geometry:
  - Vertex arrays
  - Vertex buffer objects
- Image transfers (maybe)
  - Read pixels / draw pixels
  - Color matrix
  - Pixel buffer objects

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.