

VGP351 – Week 9

⇒ Agenda:

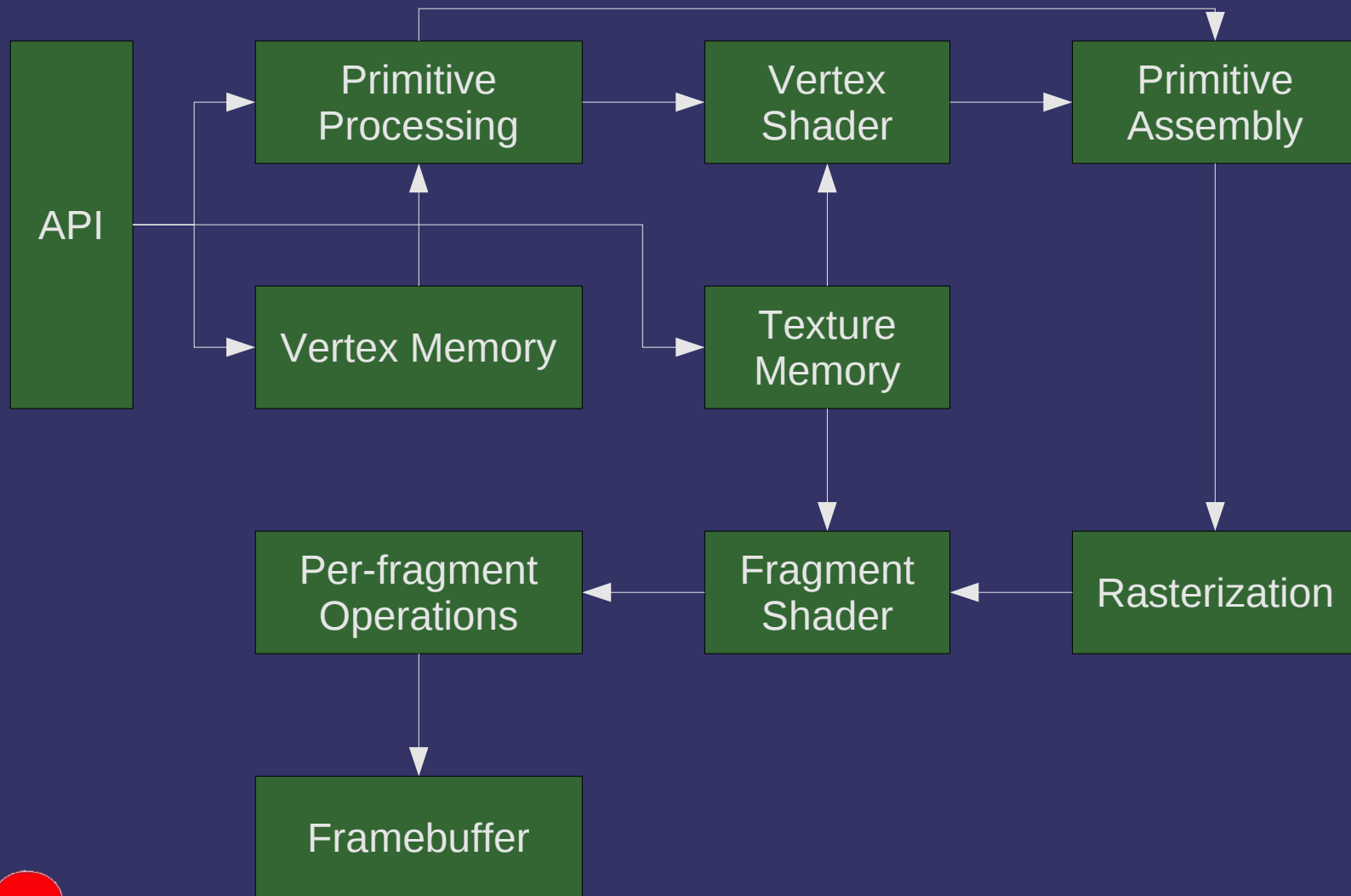
- Quiz #4
- Framebuffer blending
 - Transparency
 - Multipass rendering
- Stencil buffer
- Fog



12-March-2009

© Copyright Ian D. Romanick 2009

Graphics Pipeline



12-March-2009

© Copyright Ian D. Romanick 2009

Blending

- Last of the “per-sample” operations
 - Color output from fragment shader is combined with color already in the framebuffer
- *Many* uses!
 - Translucent / transparent objects
 - Difficult problem in the general case...objects must be rendered in the correct order and cannot intersect
 - Anti-aliasing
 - Especially useful for fonts and 2D “stroked” objects
 - 2D compositing
 - Quartz (Mac OS X), Aero (Vista), compiz (X Windows)



Multi-pass rendering

12 March 2009

© Copyright Ian D. Romanick 2009

Blending

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

Color from the fragment
shader

Color already in the
framebuffer



12-March-2009

© Copyright Ian D. Romanick 2009

Blending Function

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

Source blending function

Destination blending function

- GL_SRC_ALPHA
- GL_SRC_COLOR
- GL_DST_ALPHA
- GL_DST_COLOR
- GL_CONSTANT_COLOR
- GL_CONSTANT_ALPHA
- The above have a “one minus” form:
GL_ONE_MINUS_SRC_ALPHA
- GL_ZERO, GL_ONE
- GL_SRC_ALPHA_SATURATE
- Only available as a source factor
- $F_{src} = \min(A_s, 1 - A_d)$



12-March-2009

© Copyright Ian D. Romanick 2009

Blending Function

⇒ Blend function set with:

```
void glBlendFuncSeparate(  
    GLenum srcRGB,    GLenum dstRGB,  
    GLenum srcAlpha, GLenum dstAlpha);
```

⇒ Blend constant color set with:

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



12-March-2009

© Copyright Ian D. Romanick 2009

Blending Equation

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

Blending equation

- GL_FUNC_ADD
- GL_FUNC_SUBTRACT
- GL_FUNC_REVERSE_SUBTRACT
- GL_MIN
- GL_MAX
- Min and max equations do *not* modulate with the blend functions



12-March-2009

© Copyright Ian D. Romanick 2009

Blending Equation

⇒ Blending equation set with

```
void glBlendEquationSeparate(GLenum modeRGB,  
                             GLenum modeAlpha);
```



12-March-2009

© Copyright Ian D. Romanick 2009

Alpha Buffer

- If the desired blend modes use destination alpha, the color buffer must have alpha bits
 - As usual, ask SDL to allocate an appropriate buffer
`SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);`
 - If there is no explicit destination alpha value, the destination alpha value is implicitly 1.0



12-March-2009

© Copyright Ian D. Romanick 2009

Transparency

➤ Want to see through certain objects



Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

12-March-2009

© Copyright Ian D. Romanick 2009



Transparency

- ⇒ Transparent / translucent objects affect the appearance of objects behind them
 - Multiple levels of transparent objects accumulate additional effects



12-March-2009

© Copyright Ian D. Romanick 2009

Transparency

- ⇒ Rendering must be performed in a specific order
 - Render all non-transparent objects first
 - Render transparent objects in back-to-front order



12-March-2009

© Copyright Ian D. Romanick 2009

Alpha Test

- Sometimes transparency is used to simulate holes in objects



Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

12-March-2009

© Copyright Ian D. Romanick 2009



Alpha Test

- ⇒ *Much* faster to draw a single polygon with a texture than to draw many lines or small polygons
 - Observe that each fragment is either completely opaque ($\alpha = 1.0$) or completely transparent ($\alpha = 0.0$)



12-March-2009

© Copyright Ian D. Romanick 2009

Alpha Test

- Optimize by killing fragments with α below a certain threshold
 - Used to be performed in an extra per-sample operation call *alpha test*

```
if (calculated_color.a <= threshold)
    kill;
```



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- What do you do when the desired shading effect requires more resources than the hardware has available?



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- What do you do when the desired shading effect requires more resources than the hardware has available?
 - Use a different effect...probably with lower quality
 - Render in multiple passes



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- ⇒ Divide the shader into multiple parts
 - Partition at places where blending can combine partial results
 - Example: Perform diffuse textured pass. Configure blender to add fragment color to framebuffer. Finally, perform specular-only pass.



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- ⇒ Why do we want to render in as few passes as possible?



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- ⇒ Why do we want to render in as few passes as possible?
 - Multiple passes are almost always slower
 - Memory for each pixel must be accessed multiple times
 - Geometry must be processed multiple times
 - Usually have to change state (e.g., textures, blend modes) between passes



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- Why do we want to render in as few passes as possible?
 - Less accurate
 - Framebuffer usually only has 8 bits per component
 - Can work around this at the cost of an extra post-process pass
 - Shader math is *at least* 24-bit floating point per component



12-March-2009

© Copyright Ian D. Romanick 2009

Multi-pass Rendering

- Why do we want to render in as few passes as possible?
 - Can't always achieve the desired result
 - Doesn't work well with translucent objects
 - Can't always partition into parts that can be combined with the blender



12-March-2009

© Copyright Ian D. Romanick 2009

References

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

- Good background on general alpha blending theory

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

- Solves the ordering problem, but is complex to implement
- We'll come back to it next term :)

Peltzer, K. “Rendering Countless Blades of Waving Grass.” In GPU Gems. Ed. Randima Fernando. Upper Saddle River, NJ: Addison-Wesley Professional, April 1, 2004.

http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch07.html



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Buffer

- Extra per-pixel buffer containing integer values
 - Values in stencil buffer can be used to control drawing
 - Often interleaved with depth buffer
 - 24-bit depth and 8-bit stencil is most common
- To use stencil buffer, ask SDL to create one:

```
SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 1);
```



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Test

⇒ Drawing can be controlled via stencil test

- If the test passes, drawing proceeds
- If the test fails, the fragment is not drawn
- Enable stencil test with:

```
glEnable(GL_STENCIL_TEST);
```

- Configure stencil test with:

```
glStencilFuncSeparate(GLenum face, GLenum func,  
                     GLint ref, GLuint mask);
```



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Test

```
glStencilFuncSeparate(GLenum face, GLenum func,  
                    GLint ref, GLuint mask);
```

- `face` specifies whether front, back, or both front and back face state is set
- `func` specifies the test function: `GL_NEVER`, `GL_LESS`, `GL_LEQUAL`, `GL_GREATER`, `GL_GEQUAL`, `GL_EQUAL`, `GL_NOTEQUAL`, and `GL_ALWAYS`
- `ref` specifies the reference value for the stencil test
- `mask` specifies a mask that is ANDed with both the reference value and the stored stencil value when the test is done



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Test

- Occurs per-fragment, just like the depth test
 - Stencil test occurs *before* the depth test
 - Per-fragment operation is:
$$(\text{ref} \ \& \ \text{mask}) \ \text{op} \ (\text{stencil} \ \& \ \text{mask})$$
 - Remember: `ref`, `op`, and `mask` all depend on the polygon's facing!



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Operation

- Stencil buffer values are modified per-fragment depending on the state of the fragment:
 - Fragment failed the stencil test
 - Fragment passed the stencil test but failed the depth test
 - Fragment passed the stencil test and passed the depth test



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Operation

⇒ Eight possible operations:

- `GL_KEEP` – Keep existing value
- `GL_ZERO` – Set value to zero
- `GL_REPLACE` – Replace value with a reference value
- `GL_INCR` – Increment value, clamp to max
- `GL_INCR_WRAP` – Increment value, wrap to zero
- `GL_DECR` – Decrement value, clamp to zero
- `GL_DECR_WRAP` – Decrement value, wrap to max
- `GL_INVERT` – Bitwise inversion of value

⇒ Result is always masked with the stencil mask



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Operation

⇒ All three operations set using:

```
void glStencilOpSeparate(GLenum face,  
                        GLenum sfail, GLenum dpfail,  
                        GLenum dppass);
```

- `face` specifies whether front, back, or both front and back face state is set
- `sfail` specifies the operation for fragments that fail the stencil test
- `dpfail` specifies the operation for fragments that fail the depth test
- `dppass` specifies the operation for fragments that pass the stencil and depth tests



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Buffer

- ⇒ Clear the stencil buffer with the `GL_STENCIL_BUFFER_BIT` to `glClear`:

```
glClear(GL_STENCIL_BUFFER_BIT);
```
- If you're going to also clear the depth buffer, **always** do it at the same time as the stencil buffer!
- ⇒ Clear value is specified with `glStencilClear`



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Buffer

- Writing to bits of the stencil buffer is controlled by *another* write mask

```
void glStencilMask(GLuint mask);
```



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Buffer

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT | GL_DEPTH_BUFFER_BIT
        | GL_COLOR_BUFFER_BIT);
glEnable(GL_STENCIL_TEST);

/* Write 1 to stencil where polygon is drawn.
 */
glStencilFuncSeparate(GL_FRONT_AND_BACK,
                    GL_ALWAYS, 1, ~0);
glStencilOpSeparate(GL_FRONT_AND_BACK,
                  GL_KEEP, GL_KEEP, GL_REPLACE);
draw_some_polygon();

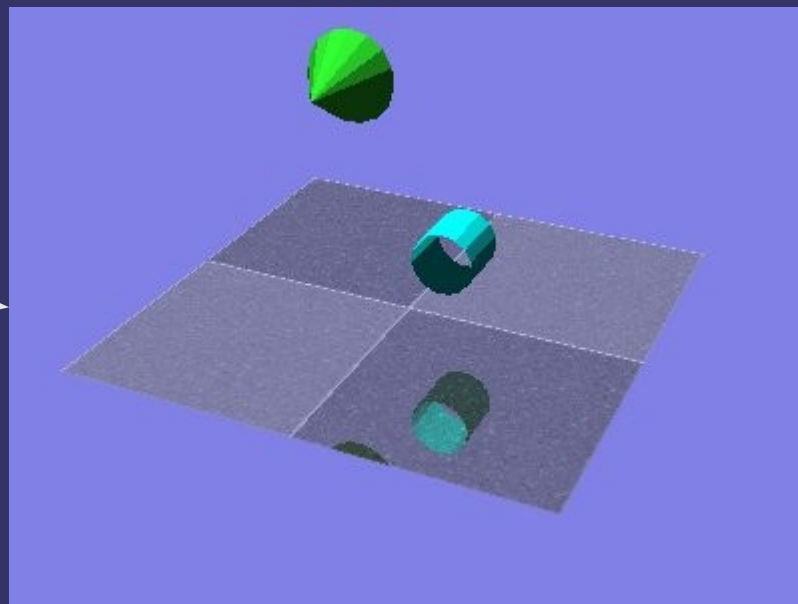
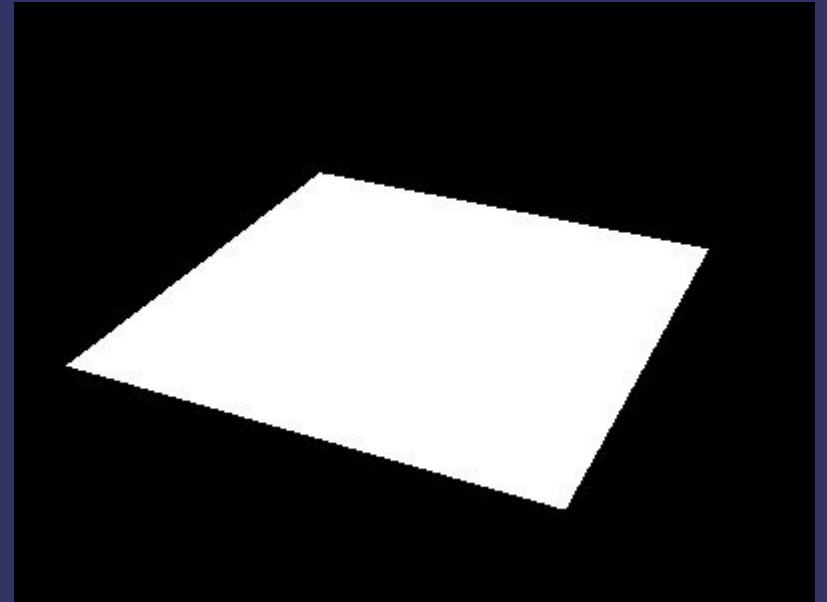
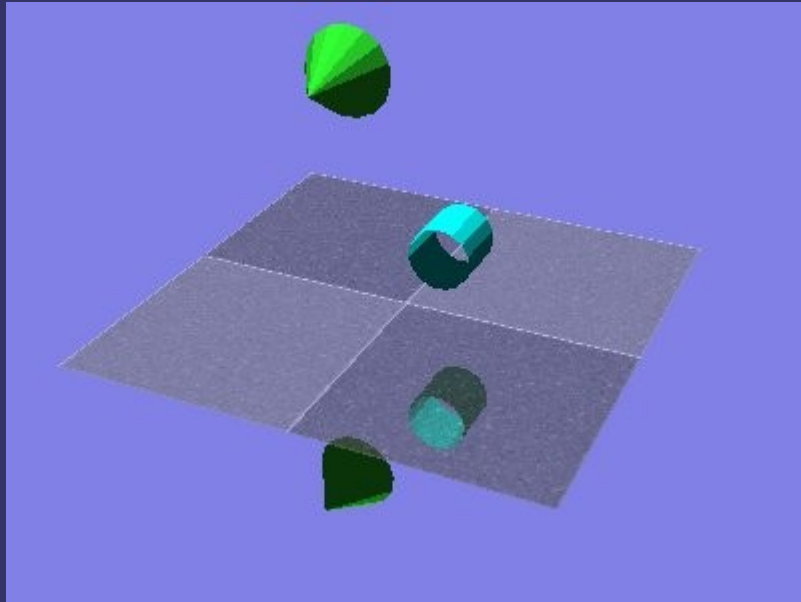
/* Draw scene only where stencil buffer is 1.
 */
glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
glStencilFuncSeparate(GL_FRONT_AND_BACK,
                    GL_EQUAL, 1, ~0);
glStencilOpSeparate(GL_FRONT_AND_BACK,
                  GL_KEEP, GL_KEEP, GL_KEEP);
draw_scene();
```

12-March-2009

© Copyright Ian D. Romanick 2009



Stencil Buffer



12-March-2009

© Copyright Ian D. Romanick 2009

Stencil Buffer



Image from Quake 3, © Copyright 1999 id Software, Inc.

12-March-2009

© Copyright Ian D. Romanick 2009

Fog

- ⇒ Typical fog... objects father away from the camera are more fog colored
 - Eventually objects disappear into the fog
 - Objects closer than some minimum distance may have no fog coloring applied



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

- ⇒ Can be used for other, related effects:
 - In dark environments, distant objects are darker
 - Underwater objects fade to the water color
 - The *only* difference is the *color* used for the fog!



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

- ⇒ Simple fog usually works in one of three modes:
 - Linear fog:

$$\frac{f_{end} - p}{f_{end} - f_{start}}$$



12-March-2009

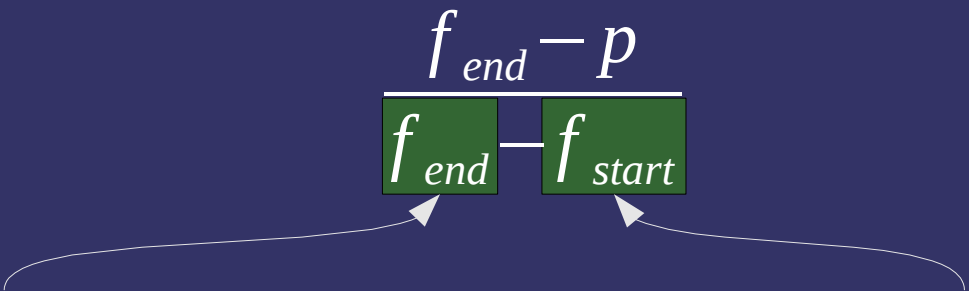
© Copyright Ian D. Romanick 2009

Fog

- Simple fog usually works in one of three modes:
 - Linear fog:

$$\frac{f_{end} - p}{f_{end} - f_{start}}$$

Distance where fog ends Distance where fog starts





12-March-2009

© Copyright Ian D. Romanick 2009

Fog

⇒ Simple fog usually works in one of three modes:

– Linear fog:

$$\frac{f_{end} - p}{f_{end} - f_{start}}$$

– Exponential fog:

$$e^{(-d \times p)}$$



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

⇒ Simple fog usually works in one of three modes:

– Linear fog:

$$\frac{f_{end} - p}{f_{end} - f_{start}}$$

– Exponential fog:

$$e^{(-d \times p)}$$

Fog density



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

⇒ Simple fog usually works in one of three modes:

– Linear fog:

$$\frac{f_{end} - p}{f_{end} - f_{start}}$$

– Exponential fog:

$$e^{(-d \times p)}$$

– Exponential-squared fog:

$$e^{(-d \times p)^2}$$



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

- Once the fog factor is calculated, use it to linearly blend between the fragment color and the fog color

$$C = F \cdot C_{\text{fragment}} + (1 - F) \cdot C_{\text{fog}}$$



12-March-2009

© Copyright Ian D. Romanick 2009

Fog

⇒ Where does p come from?

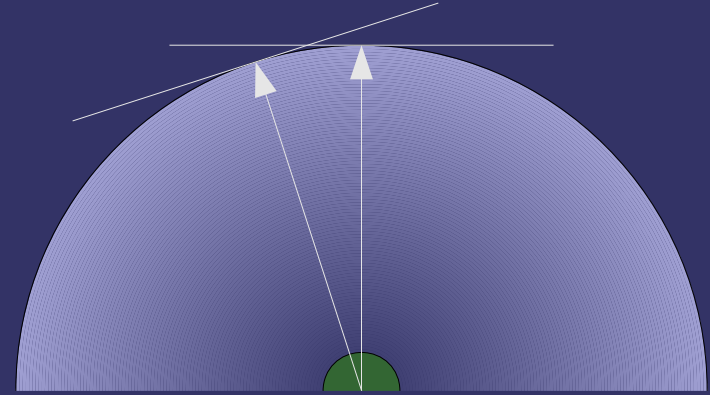


12-March-2009

© Copyright Ian D. Romanick 2009

Fog

- ⇒ Where does p come from?
 - Easy answer: eye-space Z
 - “Off center” points receive less fog than they should



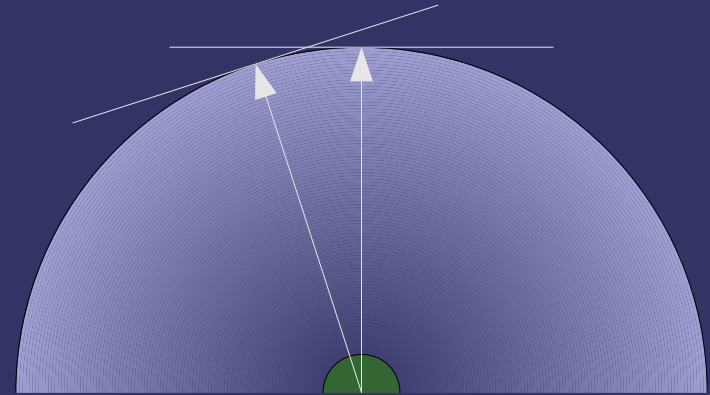
12-March-2009

© Copyright Ian D. Romanick 2009

Fog

➤ Where does p come from?

- Easy answer: eye-space Z
 - “Off center” points receive less fog than they should
- Better answer: use eye-space distance
 - More expensive to calculate
 - Still has artifacts when calculated per-vertex



12-March-2009

© Copyright Ian D. Romanick 2009

Height-based Fog

⇒ Fog factor 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



12-March-2009

© Copyright Ian D. Romanick 2009

Height-based Fog

⇒ Two components to the optical distance between the eye and the fogged point:

- Change in altitude: $\Delta y = y_{\text{point}} - y_{\text{eye}}$
- Distance in the plane: $\Delta D = \sqrt{((x_{\text{point}} - x_{\text{eye}})^2 + (z_{\text{point}} - z_{\text{eye}})^2)}$

⇒ Two important cases:

- $\Delta y = 0$: $\Delta D \times \alpha(y_{\text{point}})$

- $\Delta y \neq 0$:
$$\sqrt{1 + \left(\frac{\Delta D}{\Delta y}\right)^2} \times \int_{y_{\text{eye}}}^{y_{\text{point}}} \alpha(y) dy$$



12-March-2009

© Copyright Ian D. Romanick 2009

Height-based Fog

⇒ Two components to the optical distance between the eye and the fogged point:

- Change in altitude: $\Delta y = y_{\text{point}} - y_{\text{eye}}$
- Distance in the plane: $\Delta D = \sqrt{((x_{\text{point}} - x_{\text{eye}})^2 + (z_{\text{point}} - z_{\text{eye}})^2)}$

⇒ Two important cases:

This is the “standard” fog case!

- $\Delta y = 0$: $\Delta D \times \alpha(y_{\text{point}})$

- $\Delta y \neq 0$: $\sqrt{1 + \left(\frac{\Delta D}{\Delta y}\right)^2} \times \int_{y_{\text{eye}}}^{y_{\text{point}}} \alpha(y) dy$



12-March-2009

© Copyright Ian D. Romanick 2009

Height-based Fog

- At each index n of a look-up table, store the value:

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

- 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!



12-March-2009

© Copyright Ian D. Romanick 2009

References

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

- Older, but has some useful information and image

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

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

- Very cool example of what can be done with explicitly calculated fog coordinates. Second link has the theory behind the 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.

Nuebel, M. "Introduction to Different Fog Effects," In *ShaderX²: Introductions and Tutorials with DirectX 9*. Ed. Wolfgang Engel. Wordware, pp. 151-179, 2003.

http://www.gamedev.net/reference/programming/features/shaderx2/Introductions_and_Tutorials_with_DirectX_9.pdf



12-March-2009

© Copyright Ian D. Romanick 2009

Next week...

- ⇒ More anti-aliasing
 - AA during primitive rasterization
 - FSAA
 - Supersampling
 - Multisampling
 - Temporal AA



12-March-2009

© Copyright Ian D. Romanick 2009

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel 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.



12-March-2009

© Copyright Ian D. Romanick 2009