OpenGL Buffers and Tests

Glenn G. Chappell

CHAPPELLG@member.ams.org

U. of Alaska Fairbanks

CS 481/681 Lecture Notes

Friday, February 8, 2002

Review: The Math of Lighting

- The Phong model computes lighting separately for the ambient, diffuse, and specular components, then combines the three together.
- If the light-source color is (L_R,L_G,L_B), and the material color is (M_R,M_G,M_B), then the basic calculation is the following:
  - $R = L_R \cdot M_R$.
  - $G = L_G \cdot M_G$.
  - $B = L_B \cdot M_B$.
- For ambient, this is exactly what is done.
- For diffuse, the RGB values are multiplied by the Lambert cosine.
- For specular, the RGB values are multiplied by the cosine of the angle between the reflected ray and the viewing angle, raised to the power of the shininess (this, I believe, is Phong's primary contribution).
- To combine all the types of light, the various colors are added, and then the RGB values are clipped to [0,1].

OpenGL Buffers & Tests: Buffers

- An OpenGL buffer is an array that holds one piece of data for each pixel in the viewport.
- OpenGL has 4 types of buffers:
  - Color buffers
  - Depth buffer
  - Stencil buffer
  - Accumulation buffer
- Each buffer has an intended function; however, you may use the buffers in any way you wish.
- We will be discussing buffers for a few class meetings. The material will come primarily from chapter 10 of the red book (starts on p. 429).

OpenGL Buffers & Tests: Masking [1/2]

- Most buffers have *masks* associated with them.
  - The mask determines whether a buffer (or part of a buffer) is ever written.
- For example, the color-buffer mask is controlled by the **`glColorMask`** command.
  - This command takes 4 parameters, all **`GLboolean`**'s, representing R, G, B, and A, respectively.
  - For example,
    `glColorMask(false, true, true, true);`
    means that the R portion of the color buffer will not be changed.
  - Note: The mask affects *all* commands that would change the buffer, even **`glClear`**.

OpenGL Buffers & Tests: Masking [2/2]

- In **masking.cpp**, I define five **bool** variables: **redmask**, **greenmask**, **bluemask**, **depthmask**, **clearall**.
- Each defaults to **true** and is toggled by pressing the first letter in its name.
- The interesting part of the code is at the start of function **display**:

```
if (clearall)
{
    glColorMask(true, true, true, true);
    glDepthMask(true);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
glColorMask(redmask, greenmask, bluemask, true);
glDepthMask(depthmask);
if (!clearall)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

OpenGL Buffers & Tests: Tests [1/2]

- Associated with buffers are the OpenGL *tests*:
  - Scissor test
  - Alpha test
  - Depth test
  - Stencil test
- A *test* is an expression with a boolean value that OpenGL evaluates for every pixel to be drawn.
  - If the result is true, then the test *passes*, and the pixel is drawn.
  - Otherwise, the test *fails*, and the pixel is not drawn.

OpenGL Buffers & Tests: Tests [2/2]

- Except for the scissor test, each test is associated with a particular buffer:
  - Alpha test ↔ Color buffer (Alpha portion)
  - Depth test ↔ Depth buffer
  - Stencil test ↔ Stencil buffer
- Typically, when a test is performed, some value associated with the pixel to be drawn is compared to the data for that pixel in the buffer.

OpenGL Buffers & Tests: The Scissor Test

- The scissor test is by far the simplest of the tests.
  - It allows you to restrict drawing to a rectangular portion of the viewport.
- To enable: **`glEnable(GL_SCISSOR_TEST);`**
- Then: **`glScissor(`***x, y, width, height***`);`**
  - Parameters are as in the **`glViewport`** command.
  - (*x*,*y*) is the lower-left corner of the rectangle.
- The scissor test passes if the pixel is within the rectangle; otherwise, it fails.
- The scissor test is really just a quick, simple version of stenciling.

Review: OpenGL Buffers & Tests

- OpenGL has 4 kinds of buffers.
  - Each buffer holds a piece of data about every pixel in the viewport.
  - The kind of data depends on the kind of buffer and how it is used.
- OpenGL has 4 tests.
  - A test gives a true/false result for each pixel; if true, the test passes, and the pixel is drawn.
- Buffers and tests are associated:

Buffer	Corresponding Test
--	Scissor Test
Color Buffers	Alpha Test
Depth Buffer	Depth Test
Stencil Buffer	Stencil Test
Accumulation Buffer	--

Review: Buffer Masks

- Most buffers have *masks* associated with them.
- For example, the color-buffer mask is controlled by the **`glColorMask`** command.
  - The statement **`glColorMask(false, true, true, true);`** means that the R portion of the color buffer will not be changed.
  - Note: The mask affects *all* commands that would change the buffer, even **`glClear`**.

Review: The Scissor Test

- The *scissor test* allows you to restrict drawing to a rectangular portion of the viewport.
  - To use: enable the scissor test, and specify a rectangle with **`glScissor`**.
  - The scissor test passes if the pixel is within the rectangle; otherwise, it fails.
  - The scissor test is really just a quick, simple version of stenciling.

The Accumulation Buffer: Overview

- The most interesting of the buffers (IMHO) is the *accumulation buffer*.
  - The accumulation buffer allows you to blend together different 2-D scenes.
    - These can be renderings of 3-D scenes.
  - The accumulation buffer holds RGBA color data, just like the color buffers.
  - There are special commands that allow you to blend a color buffer with the accumulation buffer (possibly several times) and then transfer the contents of the accumulation buffer to a color buffer.
  - Allocate the accumulation buffer using **`GLUT_ACCUM`** in your **`glutInitDisplayMode`** call.

The Accumulation Buffer: Operations

- Five operations can be performed on the accumulation buffer (AB):
  - The AB can be cleared.
  - The contents of a color buffer can be multiplied by a value and then **copied** to the AB.
  - The contents of a color buffer can be multiplied by a value and then **added** to the AB.
  - An arithmetic operation (× or +) can be performed on every pixel in the AB.
  - The contents of the AB can be multiplied by a value and copied to a color buffer.
- The first operation above, clearing, is accomplished using the **`glClear`** command:

```
glClearAccum(R, G, B, A);     // like glClearColor (optional)
glClear(GL_ACCUM_BUFFER_BIT);  // Clear AB
```

- The other four operations involve the **`glAccum`** command.

The Accumulation Buffer: `glAccum` [1/2]

- **`glAccum`** takes two parameters:
  - A **`GLenum`** telling which operation to perform.
  - A **`GLfloat`** giving a relevant constant value.
- To multiply the contents of a color buffer by a value and **copy** the result to the AB:

  `glAccum(GL_LOAD,` *value*`);`

  - This uses the color buffer selected for reading. Use **`glReadBuffer`** to change this. (Generally, you do not need to worry about it.)
- To multiply the contents of a color buffer by a value and **add** the result to the AB:

  `glAccum(GL_ACCUM,` *value*`);`

The Accumulation Buffer: `glAccum` [2/2]

- To multiply the contents of the AB by a value:

  `glAccum(GL_MULT,` *value*`);`

  - There is also **`GL_ADD`**, to add instead of multiplying, but I have never seen a use for it.
- To multiply the contents of the AB by a value and copy the result to a color buffer:

  `glAccum(GL_RETURN,` *value*`);`

  - This uses the color buffer selected for drawing. Use **`glDrawBuffer`** to change this. (Generally, you do not need to worry about it.)

The Accumulation Buffer: Typical Code

```
void display()  // The display function
{
    glClear(GL_ACCUM_BUFFER_BIT);
    for (int i = 0; i < numscenes; ++i)
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        Draw scene number i here
        glAccum(GL_ACCUM, scenefraction[i]);
    }
    glAccum(GL_RETURN, 1.0);
    glutSwapBuffers();
}
```

- The values `scenefraction[i]` should be in [0,1] and should probably add up to 1.
  - Replacing "`scenefraction[i]`" with "`1.0/numscenes`" would give equal weight to all scenes being blended.
- Note how the clearing works: AB outside the loop, color & depth inside.

The Accumulation Buffer: Applications

- The AB can be used for:
  - Motion blur.
  - Fading between scenes.
  - Anti-aliasing.
  - Depth-of-field effects.
  - Soft shadows (if you know how to do shadows).
- The last three applications above are usually done with “jittering”.
  - Jittering means making repeated small perturbations to a scene.
  - Then we blend the jittered scenes together to form the finished product.
  - To do anti-aliasing and depth-of-field effects, we jitter the projection matrix; to do soft shadows, we do shadows (somehow …) and jitter the light source.
- What are some problems with using the AB?
  - AB operations are generally slow; they may be unsuitable for real-time graphics.
  - OpenGL implementations are not required to support accumulation buffers, so it might reduce the portability of code. (In practice, this does not seem to be a problem.)

Stenciling: Overview

- The stencil buffer and its associated test, the stencil test, can be used for a variety of yes/no, pass/fail-type effects.
  - The stencil buffer holds a single integer for each pixel in the viewport.
  - You can place values in the stencil buffer and then test them to determine whether to draw pixels.
  - Allocate the stencil buffer using **`GLUT_STENCIL`** in your **`glutInitDisplayMode`** call.
  - Clear the stencil buffer using
    `glClear(GL_STENCIL_BUFFER_BIT);`
    after setting the clearing value with **`glClearStencil`**.
  - Enable the stencil test using
    `glEnable(GL_STENCIL_TEST);`

Stenciling: Functions

- The two major functions used in stenciling are **`glStencilFunc`** and **`glStencilOp`**.
  - **`glStencilFunc`** determines what the stencil test does.
  - **`glStencilOp`** determines what happens to the stencil buffer if the stencil test passes or fails.
    - If the stencil test passes, then you can also have different outcomes based on the depth test.

Stenciling: `glStencilFunc`

- **`glStencilFunc`** takes three parameters:
  - A **`GLenum`** telling what comparison the stencil test will do.
  - A **`GLint`** used as a "reference value" in the stencil test.
  - A **`GLuint`** used as a mask (an "and" mask).
- Examples,
  - Stencil test passes if bit in SB is on:
    ```
    glStencilFunc(GL_EQUAL, 0x1, 0x1);
    ```
  - Stencil test passes if bit in SB is off:
    ```
    glStencilFunc(GL_NOTEQUAL, 0x1, 0x1);
    ```
  - Test passes if 20 < low 8 bits in SB:
    ```
    glStencilFunc(GL_LESS, 20, 0xff);
    ```

Stenciling: `glStencilOp`

- **`glStencilOp`** takes three parameters, all **`GLenum`**'s:
  - The operation to perform if the stencil test fails.
  - The operation to perform if the stencil test passes and the depth test fails.
  - The operation to perform if the stencil test passes and the depth test passes.
- Examples,
  - Do not modify the SB:
    `glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);`
  - Replace SB value with zero, the reference value (from **`glStencilFunc`**), or its bitwise inversion, respectively:
    `glStencilFunc(GL_ZERO, GL_REPLACE, GL_INVERT);`
  - Increment or decrement the SB value, as appropriate:
    `glStencilFunc(GL_DECR, GL_INCR, GL_INCR);`

Review: Accumulation Buffer [1/2]

- The accumulation buffer (AB) holds RGBA color data.
- It allows you to blend 2-D scenes.
- Five operations can be performed on the AB:
  - Clear AB.
  - Color buf. × value → copy to AB.
  - Color buf. × value → add to AB.
  - Arithmetic operation (× or +) on AB.
  - AB × value → copy to color buf.
- Typically:
  - Clear AB.
  - Repeat:
    - Clear color buf. And draw a scene in it.
    - Color buf. × value → add to AB.
  - Copy AB to color buf.
- Above, the values we multiply the color info by are numbers, in [0,1], whose sum is 1. Multiplying by a larger value gives that particular scene a greater weight in the final displayed image.

Review: Accumulation Buffer [2/2]

- Here is an implementation of "fade between scenes":

```
void display()  // The display function
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Draw scene 1 here
    glAccum(GL_LOAD, 1.0-fadefraction);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Draw scene 2 here
    glAccum(GL_ACCUM, fadefraction);
    glAccum(GL_RETURN, 1.0);
    glutSwapBuffers();
}
```

- The variable **`fadefraction`** should be in [0,1]. It should slowly increase from 0 to 1, changing values each time the display function is called.
- The accumulation buffer is never cleared; how can I get away with this?

Review: Stenciling [1/2]

- Stenciling involves the stencil buffer and the stencil test.
  - Remember: allocate the buffer, enable the test.
  - Clear the buffer the same way you clear any buffer.
- The two major functions used in stenciling are **`glStencilFunc`** and **`glStencilOp`**.
  - **`glStencilFunc`** determines what the stencil test does.
  - **`glStencilOp`** determines what happens to the stencil buffer if the stencil test passes or fails.
    - If the stencil test passes, then you can also have different outcomes based on the depth test.

Review: Stenciling [2/2]

- **`glStencilFunc`** takes three parameters:
  - **`GLenum`**: Which comparison the stencil test will do.
  - **`GLint`**: "Reference value" in the stencil test.
  - **`GLuint`**: Used as a mask (an "and" mask).
- **`glStencilOp`** takes three parameters:
  - **`GLenum`**: Operation to do if stencil test fails.
  - **`GLenum`**: Operation if stencil passes and depth fails.
  - **`GLenum`**: Operation if stencil passes and depth passes.

Stenciling Examples: Ordinary Stenciling

- To draw a shape in the stencil buffer:
  - Redo when viewport changes size! Code goes in the reshape function.

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_NEVER, 1, 1);
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE); // only 1st param matters
```
Draw a shape here.
```
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
```

- To use the above shape:

```
glStencilFunc(GL_EQUAL, 1, 1);
```
Draw something; it will appear only inside the above shape.
```
glStencilFunc(GL_NOTEQUAL, 1, 1);
```
Draw something; it will appear only outside the above shape.

Stenciling Examples: Odd Things to Do

- Draw each pixel at most 5 times:

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_GREATER, 5, 0xff);
glStencilOp(GL_KEEP, GL_INCR, GL_INCR);
```

- Draw each pixel successfully only on every other attempt:

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_EQUAL, 0, 1);
glStencilOp(GL_INVERT, GL_INVERT, GL_INVERT);
```

Stenciling Examples: Capping

- Here is an implementation of “capping” (see Red p. 446).
  - You are drawing a number of closed objects. You wish to make sure that the inside of these is never visible, even if the near clipping plane slices one of them.

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT |
        GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glStencilFunc(GL_ALWAYS, 1, 1);
glStencilOp(GL_INVERT, GL_INVERT, GL_INVERT);
```

Draw scene.

```
glStencilFunc(GL_EQUAL, 1, 1);
```

Draw rectangle covering entire viewport, in “capping” color.

Using Alpha: Overview

- The 4th component of RGBA color is “alpha”.
- Alpha is specified with the **`glColor*`** and **`glClearColor`** commands, as well as various lighting and material-definition commands.
- Alpha is stored in the color buffer, along with R, G, and B.
  - Alpha is always there in OpenGL’s RGB mode; so far we have not used it.
- There are two major uses of alpha:
  - Blending
    - Alpha can determine how a color to be draw is blended with the color already present at that pixel.
    - The most common application of blending is transparent objects.
  - The Alpha Test
    - Alpha can be tested, in ways similar to the stencil buffer

Using Alpha: Blending

- Blending is covered in chapter 6 of the Red Book.
  - You may also wish to read about anti-aliasing and depth-cueing ("fog") in that chapter.
- To do blending, enable it, and specify a blend function.
  - Blending is enabled with
    `glEnable(GL_BLEND);`
  - It is not necessary to allocate anything; alpha is stored in the color buffer.
  - The blending function is specified with **`glBlendFunc`**.

Using Alpha: `glBlendFunc` [1/2]

- Blending involve mixing colors based on their respective alpha values.
- A blending function blends two colors:
  - The *source* color: the color to be drawn.
  - The *destination* color: the color already present in the color buffer.
- Blending functions are specified using **`glBlendFunc`**, which takes two parameters:
  - **`GLenum`**: blending factor for the source color.
  - **`GLenum`**: blending factor for the destination color.

Using Alpha: `glBlendFunc` [2/2]

- Some possible blending factors are:
  - **`GL_ZERO`**: Multiply this color by zero (0,0,0,0).
  - **`GL_ONE`**: Multiply this color by one (1,1,1,1).
  - **`GL_SRC_ALPHA`**: Multiply this color by the source alpha.
  - **`GL_ONE_MINUS_SRC_ALPHA`**: Multiply this color by one minus the source alpha.
  - **`GL_DST_ALPHA`**: Multiply this color by the destination alpha.
  - **`GL_SRC_COLOR`** (for dest. blend factor only): Multiply this color by the source color, component by component.
- For a complete list of possible blend factors, see p. 223 of the Red Book.

Using Alpha: Applications of Blending

- What are some effects one can do with a blend function?
  - Painter's Algorithm
    - Src blend factor: 1, dest blend factor: 0 (same as no blending).
  - Transparency
    - Src bf: source alpha, dest bf: 1-source alpha.
    - Src alpha = 1: opaque, 0: invisible, between: translucent.
  - Weird Lighting Method
    - Src bf: 0, dest bf: source color.
    - Buffer holds unlit scene, source color is color of light at that point in the scene. (Alpha is ignored.)
- What difficulties are involved in using a blend function?
  - Drawing order matters. For example, when doing transparency via blending, polygons should be drawn back-to-front (use an object space HSR method).
  - Sadly, blending often gives rather bad-looking results.

Using Alpha: The Alpha Test [1/2]

- *We're back in chapter 10 now.*
- The alpha test, like the other OpenGL tests, allows you to accept or reject individual pixels.
  - As in the stencil test, a reference value is specified. The alpha value of the pixel to be drawn is compared to it.
  - To use the alpha test, enable it: **`glEnable(GL_ALPHA_TEST);`**
  - Specify an alpha test with **`glAlphaFunc`**.

Using Alpha: The Alpha Test [2/2]

- **`glAlphaFunc`** takes two parameters:
  - **`GLenum`**: What test to perform.
  - **`GLclampf`**: Reference value
    - Type is **`GLfloat`**, required to be in [0,1], that is, "clamped".
- The possible tests all compare the alpha value of the pixel to be drawn with the reference value.
  - **`GL_LESS`**: Passes if alpha to be drawn is less than the ref value.
  - **`GL_EQUAL`**: Passes if the two are equal.
  - **`GL_ALWAYS`**: Always passes.
  - Etc…
- Warning: The alpha test is done backwards from the stencil test.
  - Stencil test: REF comparison VALUE_IN_BUFFER.
  - Alpha test: VALUE_FOR_NEW_PIXEL comparison REF.