The Buffers and Operations

Fragment
  └── Scissor Test
  └── Alpha Test
  └── Stencil Test

Depth Test
  └── Blending
  └── Dithering
  └── Logical Operations

Framebuffer
Usage

• Part of the OpenGL philosophy is to provide tools without dictating their use.
  – These tools can be used in many ways.
  – Often the name (“depth buffer”) suggests the most common use, but there is nothing “wrong” with using a tool quite differently.
OpenGL Buffers

• 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.

• In order to be used, a buffer must be allocated.
  – Do this in your `glutInitDisplayMode` call.
OpenGL Tests

- Related to the 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 fragment.
  - If the result is true, then the test *passes*, and the fragment continues down the pipeline.
  - Otherwise, the test *fails*, and fragment is discarded.
  - All tests are done in the fragment-processing part of the pipeline.
- In order to have any effect, a test must be enabled.
  - Do this with `glEnable`.
Buffers & Tests Associated

- Buffers & tests

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Corresponding Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>Scissor Test</td>
</tr>
<tr>
<td>Color Buffers</td>
<td>Alpha Test</td>
</tr>
<tr>
<td>Depth Buffer</td>
<td>Depth Test</td>
</tr>
<tr>
<td>Stencil Buffer</td>
<td>Stencil Test</td>
</tr>
<tr>
<td>Accumulation Buffer</td>
<td>--</td>
</tr>
</tbody>
</table>

- Remember: Allocate buffers; enable tests!
Most buffers have *masks* associated with them. If flag is GL_FALSE, buffer writing is disabled.

- The mask determines whether a buffer (or part of a buffer) is ever written.
- For example,
  ```c
  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`.

```c
void glIndexMask(GLuint mask);
void glColorMask(GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha);
void glDepthMask(GLboolean flag);
void glStencilMask(GLuint mask);
```
Buffers Clearing

- The buffers to clear are specified by bitwise-or’ing together. For example,
  ```c
  glClear(GL_COLOR_BUFFER_BIT |
          GL_DEPTH_BUFFER_BIT |
          GL_STENCIL_BUFFER_BIT |
          GL_ACCUM_BUFFER_BIT);
  ```

- The value to store in each pixel of a buffer is set by a separate command for each buffer:
  ```c
  glClearColor
  glClearDepth
  glClearStencil
  glClearAccum
  ```
- `glGetIntegerv()` to query your OpenGL system about per-pixel buffer storage for a particular buffer

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_RED_BITS,</td>
<td>Number of bits per R, G, B, or A component in the color buffers</td>
</tr>
<tr>
<td>GL_GREEN_BITS,</td>
<td></td>
</tr>
<tr>
<td>GL_BLUE_BITS,</td>
<td></td>
</tr>
<tr>
<td>GL_ALPHA_BITS</td>
<td></td>
</tr>
<tr>
<td>GL_DEPTH_BITS</td>
<td>No of bits per pixel in the depth buffer</td>
</tr>
<tr>
<td>GL_STENCIL_BITS</td>
<td>No of bits per pixel in the stencil buffer</td>
</tr>
<tr>
<td>GL_ACCUM_RED_BITS,</td>
<td>No of bits per R, G, B, or A component in the accumulation buffer</td>
</tr>
<tr>
<td>GL_ACCUM_GREEN_BITS,</td>
<td></td>
</tr>
<tr>
<td>GL_ACCUM_BLUE_BITS,</td>
<td></td>
</tr>
<tr>
<td>GL_ACCUM ALPHA_BITS</td>
<td></td>
</tr>
</tbody>
</table>
• Color Buffers (Framebuffers)
  • Stereoscopic viewing has left and right color buffers for the left and right stereo images.
  • Double-buffered systems have front & back buffers.
  • Up to four optional, non-displayable auxiliary color buffers can also be supported.

• Stencil Buffer
  • One use for the stencil buffer is to restrict drawing to certain portions of the screen, just as a cardboard stencil can be used with a can of spray paint to make fairly precise painted images.

  • You can store an image of the windshield’s shape in the stencil buffer, and then draw the entire scene.
• Accumulation Buffer
  • The accumulation buffer holds RGBA color data just like the color buffers do in RGBA mode.
  • It’s typically used for accumulating a series of images into a final, composite image.
  • You can perform operations like scene antialiasing by supersampling an image and then averaging the samples to produce the values that are finally painted into the pixels of the color buffers.
  • You don’t draw directly into the accumulation buffer; accumulation operations are always performed in rectangular blocks, usually transfers of data to or from a color buffer.
Testing and Operating on Fragments

• Scissor Test
  • If a fragment lies inside the rectangle, it passes the scissor test.
  • `void glScissor(GLint x, GLint y, GLsizei width, GLsizei height);` Sets the location and size of the scissor rectangle. The parameters define the lower left corner `(x, y)`, and the width and height of the rectangle.
  • Pixels that lie inside the rectangle pass the scissor test. Scissoring is enabled and disabled by passing `GL_SCISSOR` to `glEnable()` and `glDisable()`.
  • The scissor test is just a version of a stencil test using a rectangular region of the screen. It’s fairly easy to create a blindingly fast hardware implementation of scissoring, while a given system might be much slower at stenciling -- perhaps because the stenciling is performed in software.
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.
• **Alpha Test**

In RGBA mode, the alpha test allows you to accept or reject a fragment based on its alpha value.

```c
void glAlphaFunc(GLenum func, GLclampf ref);
```

Sets the reference value and comparison function for the alpha test. The reference value `ref` is clamped to be between 0 and 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_NEVER</td>
<td>Never accept the fragment</td>
</tr>
<tr>
<td>GL_ALWAYS</td>
<td>Always accept the fragment</td>
</tr>
<tr>
<td>GL_LESS</td>
<td>Accept fragment if fragment alpha &lt; reference alpha</td>
</tr>
<tr>
<td>GL_LEQUAL</td>
<td>Accept fragment if fragment alpha &lt;= reference alpha</td>
</tr>
<tr>
<td>GL_EQUAL</td>
<td>Accept fragment if fragment alpha = reference alpha</td>
</tr>
<tr>
<td>GL_GEQUAL</td>
<td>Accept fragment if fragment alpha &gt;= reference alpha</td>
</tr>
<tr>
<td>GL_GREATER</td>
<td>Accept fragment if fragment alpha &gt; reference alpha</td>
</tr>
<tr>
<td>GL_NOTEQUAL</td>
<td>Accept fragment if fragment alpha != reference alpha</td>
</tr>
</tbody>
</table>
Alpha Test

• Reject pixels based on their alpha value
  • `glAlphaFunc( func, value )`
  • `glEnable( GL_ALPHA_TEST )`
  – use alpha as a mask in textures
void glStencilFunc(GLenum func, GLint ref, GLuint mask);

• The ref value is compared to the value in the stencil buffer using the comparison func, but the comparison applies only to those bits where the corresponding bits of the mask are 1.

• The function can be GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, or GL_NOTEQUAL.

• The stencil test is enabled and disabled by passing GL_STENCIL_TEST to glEnable() and glDisable().

void glStencilOp(GLenum fail, GLenum zfail, GLenum zpass);

Specifies how the data in the stencil buffer is modified when a fragment passes or fails the stencil test. The fail, zfail, and zpass can be GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, or GL_INVERT. They correspond to keeping the current value, replacing it with zero, replacing it with the reference value, incrementing it, decrementing it, or bitwise-inverting it.
Controlling Stencil Buffer

- `glStencilFunc(func, ref, mask)`
  - compare value in buffer with `ref` using `func`
  - only applied for bits in `mask` which are 1
  - `func` is one of standard comparison functions
- `glStencilOp(fail, zfail, zpass)`
  - Allows changes in stencil buffer based on passing or failing stencil and depth tests: `GL_KEEP`, `GL_INCR`
Creating a Mask

- `glInitDisplayMode(...|GLUT_STENCIL|... );`
- `glEnable( GL_STENCIL_TEST );`
- `glClearStencil( 0x0 );`
- `glStencilFunc( GL_ALWAYS, 0x1, 0x1 );`
- `glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );`
- `draw mask`
Using Stencil Mask

- Draw objects where stencil = 1
  - `glStencilFunc(GL_EQUAL, 0x1, 0x1)`

- Draw objects where stencil != 1
  - `glStencilFunc(GL_NOTEQUAL, 0x1, 0x1);`
  - `glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);`
• Depth Test

• void glDepthFunc(GLenum func);
• Sets the comparison function for the depth test. The value for func must be GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, or GL_NOTEQUAL.
• An incoming fragment passes the depth test if its z value has the specified relation to the value already stored in the depth buffer. The default is GL_LESS. In this case, the z value represents the distance from the object to the viewpoint.
The Accumulation Buffer

• Images is generated in one of the standard color buffers, and these are accumulated, one at a time, into the accumulation buffer.

• When the accumulation is finished, the result is copied back into a color buffer for viewing.

• To reduce rounding errors, the accumulation buffer may have higher precision (more bits per color) than the standard color buffers.

• A photographer typically creates a multiple exposure by taking several pictures of the same scene without advancing the film. If anything in the scene moves, that object appears blurred.
void glAccum(GLenum op, GLfloat value);

Controls the accumulation buffer. The op parameter selects the operation, and value is a number to be used in that operation.

The possible operations are GL_ACCUM, GL_LOAD, GL_RETURN, GL_ADD, and GL_MULT:

- **GL_ACCUM** reads each pixel from the buffer currently selected for reading, multiplies the R, G, B, and alpha values by value, and adds the result to the accumulation buffer.

- **GL_LOAD** does the same thing, except that the values replace those in the accumulation buffer rather than being added to them.

- **GL_RETURN** takes values from the accumulation buffer, multiplies them by value, and places the result in the color buffer(s) enabled for writing.

- **GL_ADD** and **GL_MULT** simply add or multiply the value of each pixel in the accumulation buffer by value, and then return it to the accumulation buffer. For **GL_MULT**, value is clamped to be in the range \([-1.0,1.0]\). For **GL_ADD**, no clamping occurs.
• Scene Antialiasing

First clear the accumulation buffer and enable the front buffer for reading and writing. Then loop several times (say, n) through code that draws the image in a slightly different position, accumulating the data with

```glAccum(GL_ACCUM, 1.0/n);
```

and finally calling

```glAccum(GL_RETURN, 1.0);
```

To decide what n should be, you need to trade off speed (the more times you draw the scene, the longer it takes to obtain the final image) and quality (the more times you draw the scene, the smoother it gets, until you make maximum use of the accumulation buffer’s resolution).

• Motion Blur

![Motion Blur Diagram]

Copyright @ 2001 by Jim X. Chen:

jchen@cs.gmu.edu
• Suppose you want to make a motion-blurred image extending over a small interval of time.

• Instead of spatially jittering the images, jitter them temporally. The entire scene can be made successively dimmer by calling

   `glAccum (GL_MULT, decayFactor);`

   where `decayFactor` is a number between 0.0 & 1.0.

• Smaller numbers for `decayFactor` cause the object to appear to be moving faster. You can transfer the completed scene with the object’s current position and “vapor trail” of previous positions from the accumulation buffer to the standard color buffer with

   `glAccum (GL_RETURN, 1.0);`

• The image looks correct even if the items move at different speeds, or if some of them are accelerated. As before, the more jitter points you use, the better the final image.

• You can combine motion blur with antialiasing by jittering in both the spatial and temporal domains, but you pay for higher quality with longer rendering times.
• Depth of Field

• A photograph made with a camera is in perfect focus only for items lying on a single plane a certain distance from the film. The farther an item is from this plane, the more out of focus it is.

• The depth of field for a camera is a region about the plane of perfect focus where items are out of focus by a small enough amount.

• The accumulation buffer can be used to approximate what you would see in a photograph where items are more and more blurred as their distance from a plane of perfect focus increases.

• It isn’t an exact simulation of the effects produced in a camera, but the result looks similar.

• To achieve this result, draw the scene repeatedly using calls with different arg values to glFrustum().

• Choose the arguments so that the position of the viewpoint varies slightly around its true position and so that each frustum shares a common rectangle that lies in the plane of perfect focus.
• Soft Shadows

To accumulate soft shadows due to multiple light sources, render the shadows with one light turned on at a time, and accumulate them together.
Dithering

- `glEnable( GL_DITHER )`
- Dither colors for better looking results
  - Used to simulate more available colors
Logical Operations on Pixels

• Combine pixels using bitwise logical operations
  - `glLogicOp(mode)`
  
  – Common modes
    • GL_XOR
    • GL_AND