Advanced OpenGL

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd
From last time …

- **Flexible shading**
  - shading networks
  - shading languages

- **GLSL**
  - language features
  - interfacing shaders with the application
Topics for today

- Extensions
- Performance optimizations
  - Display lists
  - Triangle strips and fans
  - Vertex arrays / vertex buffer objects
- Multipass rendering
  - Render-to-texture
- Accumulation buffer
- Back end operations
  - Scissor test, stencil test, alpha test, blending, etc.
OpenGL extensions

- **OpenGL** has a flexible extension mechanism

- Tokens and functions from extension have special suffixes
  - _NV or _ATI - vendor specific
  - _EXT - general extensions
  - _ARB - approved by architecture review board

- **ARB** extensions are then folded into later versions of **OpenGL**.
  - Features from later versions are accessed through extension mechanism
GLEW

- The OpenGL Extension Wrangler (GLEW) hides details of initializing extensions
  - http://glew.sourceforge.net/

```c
glewInit();
if (GLEW_EXT_framebuffer_object)
{
   /* It is safe to use the EXT_framebuffer_object extension here. */
}

if (GLEW_VERSION_2_0)
{
   /* It is safe to use version 2.0 functions. */
   glAttachShader(programID, codeStr);
}
```
Immediate mode versus display lists

Immediate Mode

- Polynomial Evaluator
- Per Vertex Operations & Primitive Assembly
  - Rasterization
  - Per Fragment Operations
  - Frame Buffer

Display List
- CP

Pixel Operations
- Texture Memory
Display lists

Creating a display list

```python
def init(void):
global id
id = glGenLists( 1 )
gNewList( id, GL_COMPILE )
# various OpenGL routines
gEndList()
```

Call a created list

```python
def display():
gCallList( id )
```

gNewList also accepts the constant
GL_COMPILE_AND_EXECUTE, which both creates
and executes a display list.

If a new list is created with the same identifying
number as an existing display list, the old list is
replaced with the new calls.
An example

cow = None
def drawCow(color):
    global cow, cowID
    if (cow == None):
        cow = WaveFrontOBJ('cow.obj')
        cowID = glGenLists(1)
        glNewList(cowID, GL_COMPILE)
        glPushMatrix()
        glScaled(0.8,0.8,0.8)
        glTranslated(0,-cow.min.y,0)
        cow.draw()
        glPopMatrix()
        glEndList()
    glEnable(GL_LIGHTING)
    glMaterialfv(GL_FRONT, GL_AMBIENT, color)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, color)
    glCallList(cowID)

def drawScene():
    drawFloor()
    glTranslated(0,0,6)
    drawCow([0.6, 0.0, 0.8, 1.0])
    glTranslated(0,0,6)
    drawCow([0.8, 0.6, 0.0, 1.0])
    glTranslated(-12,0,-6)
    drawCow([0.0, 0.6, 0.8, 1.0])
    ...
Display lists

- Not all OpenGL routines can be stored in display lists
- Display lists can call other display lists
- Display lists are not editable, but you can fake it
  - make a list (A) which calls other lists (B, C, and D)
  - delete and replace B, C, and D, as needed
- Display lists can provide a significant speed-ups over immediate mode
Consider model of a car

- Create display list for chassis
- Create display list for wheel

```c
glNewList( CAR, GL_COMPILE )
glCallList( CHASSIS )
glTranslatef( ... )
glCallList( WHEEL )
glTranslatef( ... )
glCallList( WHEEL )
...
glEndList()
```
Efficient primitive representations

- Triangle strips and fans avoid specifying redundant vertex information
- Each triangle specified with about 1 vertex on average

Triangle Strip

Triangle Fan
Vertex arrays

- Vertex arrays allow for more efficient memory access
- Vertex arrays can specify a batch of vertices in a single call
- Indexed vertex arrays avoid specifying redundant vertex information
- Can specify vertex data in a wide variety of formats
- Vertex arrays can be stored on the GPU in Vertex Buffer Objects (VBOs)
OpenGL Vertex Arrays

- Specify locations of the vertex attribute arrays
- Enable the arrays and render
- Render with `glDrawArrays` or `glDrawElements()`

```c
GLfloat coords[] = { /* vertex coordinates */ }
GLfloat colors[] = { /* vertex colors */ }

glVertexPointer( 3, GL_FLOAT, 0, coords )
glColorPointer( 4, GL_FLOAT, 0, colors )
glEnableClientState( GL_VERTEX_ARRAY )
glEnableClientState( GL_COLOR_ARRAY )
glDrawArrays( GL_TRIANGLES, 0, numVerts )
-or-
glDrawElements( GL_TRIANGLES, 20, GL_SHORT, indices )
```

- Processes each vertex sequentially
- Allows random access to vertices, thus supporting sharing
Multipass rendering

- Breaks up complex rendering into more than one rendering pass.
- Can combine passes into the frame buffer with blending.
- Can use multiple passes to store results in textures that can be used in subsequent rendering passes:
  - Dynamic shadow maps
  - Dynamic cube maps
  - GPGPU
Render-to-texture

- `glCopyTexImage2D()` copies the frame buffer to a texture – but slow

- The Frame Buffer Object (FBO) extension allows rendering directly to a texture
  - Allows you to attach textures to use as color and depth buffers
  - Buffers have to be the same size
  - Not all possible combinations are allowed
  - Can also attach render buffers in place of textures

- Aaron Lefohn’s FBO class provides encapsulation of details
  - http://sourceforge.net/projects/gpgpu/
// initialize textures
glBindTexture( GL_TEXTURE_2D, colorTexID );
... set texture parameters ...
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, fboW, fboH, 0,
               GL_RGBA, GL_UNSIGNED_BYTE, NULL );

glBindTexture( GL_TEXTURE_2D, depthTexID );
... set texture parameters ...
glTexImage2D( GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, fboW, fboH,
               GL_DEPTH_COMPONENT, GL_UNSIGNED_INT, NULL );

// initialize fbo
fbo = new FramebufferObject();
fbo->Bind();
fbo->AttachTexture( GL_TEXTURE_2D, colorTexID,
                    GL_COLOR0_ATTACHMENT_EXT );
fbo->AttachTexture( GL_TEXTURE_2D, depthTexID,
                    GL_DEPTH_ATTACHMENT_EXT );
if( !fbo->IsValid() )
    printf("FBO format not supported.\n" );
fbo->Disable();
Using an FBO

```cpp
fbo->Bind();
glClearColor(1,1,1,1);
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

// setup the viewing transformations

// draw a rotating wire torus
float t = glutGet( GLUT_ELAPSED_TIME ) / 1000.0;
glRotatef( 20*t, 0, 1, 0 );
glRotatef( 10*t, 1, 0, 0 );
gLineWidth( 3 );
glColor3f(0,0,0);
glutWireTorus( 0.5, 1, 20, 20 );
gLineWidth( 1 );

fbo->Disable();
```
OpenGL accumulation buffer

- Problems of compositing into color buffers
  - limited color resolution due to clamping and limited precision

- Accumulation buffer has higher precision
  - accumulate multiple rendering passes into accumulation buffer
  - transfer results to frame buffer

- Applications
  - Explicit “Do-it-yourself” Compositing
  - Full Scene Antialiasing
  - Depth of Field
  - Motion Blur
  - Filtering
OpenGL accumulation buffer

- `glAccum( op, value)`

- **Functions performed for various values of op:**
  - `GL_LOAD`: $AB = FB \times value$
  - `GL_ACCUM`: $AB += FB \times value$
  - `GL_ADD`: $AB += value$
  - `GL_MUL`: $AB *= value$
  - `GL_RETURN`: $FB = AB \times value$

AB = accumulation buffer
FB = frame buffer
Each time we move the viewer, the image shifts

- Different aliasing artifacts in each image
- Averaging images using accumulation buffer averages out these artifacts
samples = 4
for i in xrange(samples*samples):
    dx = ((i % samples) - 0.5*(samples-1))/float(samples)
    dy = ((i / samples) - 0.5*(samples-1))/float(samples)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    drawScene((dx,dy))
    if (i == 0):
        glAccum(GL_LOAD, 1/float(samples*samples))
    else:
        glAccum(GL_ACCUM, 1/float(samples*samples))
    glAccum(GL_RETURN, 1)

def drawScene(dx,dy):
    glMatrixMode( GL_PROJECTION)
    glLoadIdentity()
    glTranslate( dx, dy )
    glPerspective( fov, aspect, znear, zfar)
    ...
Full Scene Antialiasing
Depth of Field

- Jitter the viewpoint in such a way as to leave the focal plane unchanged
- accumulate each rendered view
Getting to the framebuffer

- Scissor Test
- Alpha Test
- Stencil Test
- Depth Test
- Blending
- Dithering
- Logical Operations

CPU
- DL
- Texture
- Per Vertex
- Poly.
- Frag
- Raster
- Pixel
- FB
Scissor Test

- **glScissor( x, y, w, h )**
  - Additional clipping test
  - any fragments outside of box are clipped
  - useful for updating a small section of a viewport
  - enable with `glEnable( GL_SCISSOR_TEST)`

![Scissor Box](image)

![Viewport](image)
Alpha Test

- `glAlphaFunc( func, value )`
  - Reject pixels based on their alpha value
  - Enable with `glEnable( GL_ALPHA_TEST )`
  - use alpha as a mask in textures
  - `func` can be one of the following:

```plaintext
GL_NEVER  GL_LESS
GL_EQUAL  GL_LEQUAL
GL_GREATER GL_NOTEQUAL
GL_GEQUAL  GL_ALWAYS
```
Stencil Buffer

- Stencil buffer associates counter with each pixel
  - updated depending on the result of the stencil and depth tests

- The stencil test rejects pixels based on the contents of the stencil buffer

- Lots of creative uses
  - Irregular shaped rendering regions
  - Screen-door transparency
  - Counting the number of times a given pixel is touched (depth complexity)
  - CSG operations (Intersection, and Difference)
  - Shadows
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
    - GL_NEVER, GL_LESS, GL_LEQUAL, GL_GREATER, GL_GEQUAL, GL_EQUAL,
    - GL_NOTEQUAL, GL_ALWAYS

- **glStencilOp( stencilFail, zfail, zpass )**
  - Modify stencil buffer based on stencil and depth tests:
    - GL_KEEP, GLZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT
Creating a Mask

- Setup mask
  ```
  glEnable( GL_STENCIL_TEST );
  glClearStencil( 0x0);
  glStencilFunc( GL_ALWAYS, 0x1, 0x1 );
  glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
  // draw mask
  ```

- draw objects where stencil != 1
  ```
  glStencilFunc( GL_EQUAL, 0x1, 0x1 )
  // draw objects
  ```

- draw objects where stencil = 1
  ```
  glStencilFunc( GL_NOT_EQUAL, 0x1, 0x1 )
  glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP )
  // draw objects
Alpha blending

- Combines fragment with contents of the frame buffer

- Uses
  - simulate translucent objects
    - transparent objects must be rendered after opaque ones and in back to front order
  - composite images
  - antialiasing

- Enabled with `glEnable(GL_BLEND)`
  - alpha component ignored when blending disabled
Alpha Blending Modes

- `glBlendFunc( src, dst )`
- `src` and `dst` can be one of the following:
  - `GL_ONE`
  - `GL_ZERO`
  - `GL_SRC_ALPHA`
  - `GL_ONE_MINUS_SRC_ALPHA`

$$\tilde{C}_r = src \, \tilde{C}_f + dst \, \tilde{C}_p$$
Dithering

- Dither colors for better looking results
  - Used to simulate more available colors
  - Particularly when using 16-bit colors
  - Often used by gamers to double bandwidth
Logical Operations on Pixels

- **glLogicOp( mode )**

  - Combine pixels using *bitwise* logical operations
    - **Common modes**
      - GL_XOR – useful for rubber-banding style selection
      - GL_AND – useful for masking off bits

    - **All 16 possible boolean functions of source and destination**
      - GL_CLEAR        GL_SET        GL_COPY,
      - GL_COPY_INVERTED GL_NOOP       GL_INVERT
      - GL_AND          GL_NAND       GL_OR
      - GL_NOR          GL_XOR        GL_AND_INVERTED
      - GL_AND_REVERSE  GL_EQUIV      GL_OR_REVERSE
      - GL_OR_INVERTED
Antialiasing

- `glEnable( mode )`
  - `GL_POINT_SMOOTH`
  - `GL_LINE_SMOOTH`
  - `GL_POLYGON_SMOOTH`

  ° alpha value determined by pre-computing sub-pixel coverage around object silhouettes
  ° available in both RGBA and colormap modes
Fog

- `glFog[fvd]( property, value )`

- Depth Cueing
  - Specify a range for a linear fog ramp
    - `GL_FOG_LINEAR`

- Environmental effects
  - Simulate more realistic fog
    - `GL_FOG_EXP`
    - `GL_FOG_EXP2`

- Other Properties
  - `GL_FOG_COLOR`, `GL_FOG_START`, `GL_FOG_END`
Assignment #3
Next time

- Visibility computations