Open GL

A Crash Course

Part 2

Selected Topics:

- Display Lists
- Alpha blending
- Shadows (projection and blending)
- Mirrors (stencil buffer)
- Scene Antialiasing (accumulation buffer)
- Texturing

Display Lists

Why are they faster?

- Static, pre-cached list
- Pre-calculation (states, matrices)
- Use hardware most efficiently (texture memory
- Maintained server side

Typical applications

- Encapsulate repeated state changes
- Store repeated models and transforms
- Texturing definitions

General Usage

```
listIndex = glGenLists(1);
glNewList(listIndex,GL_COMPILE);
   //GL commands to be stored
glEndList();
…
glCallList(listIndex);
```

Advanced Example

- As with direct OpenGL calls, status is commulative
- We might want to save/restore state

```
glNewList(listIndex,GL_COMPILE);
    glPushMatrix();
    glPushAttrib(GL_CURRENT_BIT);
    glColor3f(1.0,0.0,0.0);
    glTranslatef(1.5,0,0);
    glBegin(GL_POLYGON);
        glVertex3f(1.0,0.0,0.0);
        glVertex3f(1.0,1.0,0.0);
        glVertex3f(1.0,0.0,1.0);
    glEnd();
    glPopAttrib();
    glPopMatrix();
glEndList();
```

Other notes

- Can call lists from within lists
- Worth using for non repeated complex geometry, etc.

Alpha Blending

Some Uses

- Make transparent objects
- Drawing shadows
- Billboarding

Implementation

```
glEnable(GL_BLEND)
glDisable(GL_BLEND)
glBlendFunc(src_factor,dst_factor)
```

Source/Dest Color: $(R_S, G_S, B_S, A_S) / (R_D, G_D, B_D, A_D)$
Source/Dest Factor: $(S_R, S_G, S_B, S_A) / (D_R, D_G, D_B, D_A)$

Gives: $(R_S S_R + R_D D_R,\ G_S S_G + G_D D_G,\ B_S S_B + B_D D_B,\ A_S S_A + A_D D_A)$

Factor Type	Computed Factor
GL_ZERO	$(0,0,0,0)$
GL_ONE	$(1,1,1,1)$
GL_SRC_ALPHA	(A_S, A_S, A_S, A_S)
GL_ONE_MINUS_SRC_ALPHA	$(1,1,1,1) - (A_S, A_S, A_S, A_S)$
GL_DST_ALPHA	(A_D, A_D, A_D, A_D)
GL_ONE_MINUS_DST_ALPHA	$(1,1,1,1) - (A_D, A_D, A_D, A_D)$
GL_DST_COLOR (src)	(R_D, G_D, B_D, A_D)
GL_SRC_COLOR (dst)	(R_S, G_S, B_S, A_S)
GL_SRC_ALPHA_SATURATE (src)	$(f,f,f,1); f = \min(A_S, 1-A_D)$

Some Examples

Solid background with a 20% opaque image in front.

- Draw background using GL_ONE (source) and GL_ZERO (dest)
- Draw the transparent image (with alpha of 0.20) using GL_SRC_ALPHA (source) and GL_ONE_MINUS_SRC_ALPHA (dest)

Mix three images equally

- Draw each image (with alpha of 0.33333) using GL_SRC_ALPHA (source) and GL_ONE (dest)

Transparent red color filter

- Draw background using GL_ONE (source) and GL_ZERO (dest)
- Draw filter image (1.0,0.1,0.1,1.0) using GL_ZERO (source) and GL_SRC_COLOR (destination)

3-D Issues

- Order to draw objects in?
- Depth buffer testing and setting?
- Multiple transparent objects?

Solutions

- Turn on depth testing

```
glEnable(GL_DEPTH_TEST)
```

- Draw all opaque objects first
- Make depth buffer read only

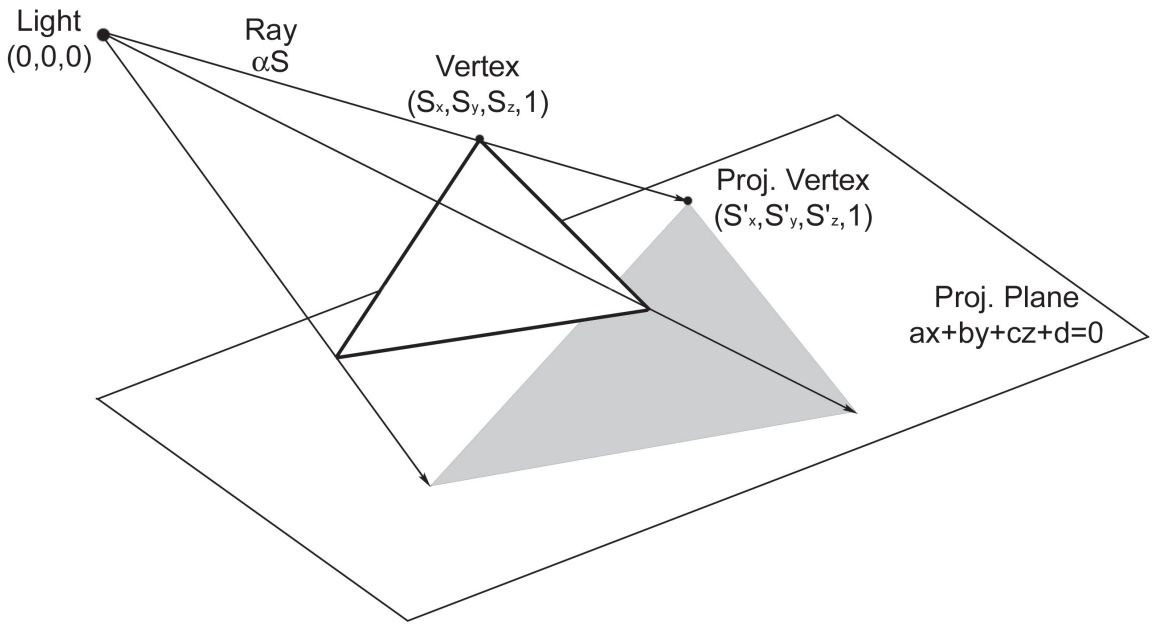
```
glDepthMask(GL_FLASE)
```

- Draw transparent objects in order (back to front)
- Make depth buffer read/write

```
glDepthMask(GL_TRUE)
```

Shadows:

- OpenGL has no built-in shadows
- There are several techniques
- Easiest: Redraw each shadowed primitive with a projection matrix in a ‘shadow’ color



- Cast a ray from light, through vertex S, and intersect the shadow plane, S’
- Redraw object in a ‘shadow’ color using a matrix that maps S to S’
- Repeat for each light and each shadow plane

Mathematical Details

$$S = (s_x, s_y, s_z)$$

$$ax + by + cz + d = 0$$

$$\alpha S = \alpha(s_x, s_y, s_z)$$

$$\alpha(a \cdot s_x + b \cdot s_y + c \cdot s_z) + d = 0$$

$$\alpha = \frac{-d}{a \cdot s_x + b \cdot s_y + c \cdot s_z}$$

$$S' = \frac{-dS}{a \cdot s_x + b \cdot s_y + c \cdot s_z}$$

$$\begin{bmatrix} s'_x \\ s'_y \\ s'_z \\ s'_w \end{bmatrix} = \begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{bmatrix} \begin{bmatrix} s_x \\ s_y \\ s_z \\ s_w \end{bmatrix} = \begin{bmatrix} -ds_x \\ -ds_y \\ -ds_z \\ ax + by + cz \end{bmatrix} \rightarrow \begin{bmatrix} \frac{-ds_x}{ax + by + cz} \\ \frac{-ds_y}{ax + by + cz} \\ \frac{-ds_z}{ax + by + cz} \\ 1 \end{bmatrix}$$

Pseudo Code

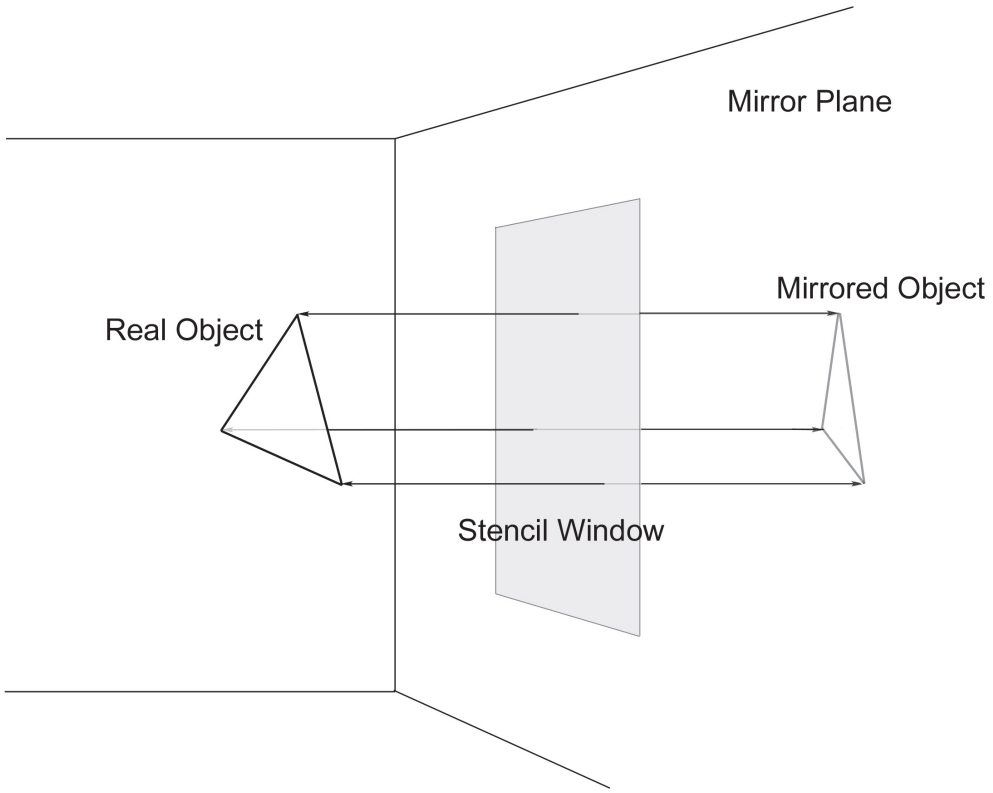
```
Set object material/color
Draw object

Turn on alpha blending
Change depth buffer to read only
Change material to black (with alpha)

Push Matrix
Translate light away from origin
Apply shadow matrix
Translate light to origin
Redraw object
Pop Matrix

Turn of alpha blending
Restore depth buffer to read/write
```

Details

- Use `glMultMatrix`
- Repeat for each light source
- Repeat for each shadow plane

Mirrors

- OpenGL has no built-in reflection capabilities
- There are several techniques
- Easiest: Reflect the world around the mirror plane and redraw it!



- Add transform to reflect about mirror plane
- Draw the world
- Remove transform and draw world again

Issues

Correct lighting?

- Must define lights after adding mirror transform
- Must redefine lights after removing transform

Non-infinite mirror?

- Draw the mirror surface into the stencil buffer
- Redraw the mirrored world through the stencil buffer

Not a fully reflecting surface?

- Draw stenciled, mirrored world
- Alpha blend mirror surface on top

Stencil buffer

- Used to restrict drawing to certain portions of the screen
- Two properties can be configured to produce various operations:
  - The pass/fail criteria
  - How to modify on pass/fail
- In our case:
  - Setup to always pass test, and put 1 into pixels where the test occurs
  - Draw mirror surface
  - Setup to pass where stencil buffer is 1, and not modify on pass/fail
  - Draw mirrored world

Clearing

```
glClearStencil(0)
glClear(GL_STENCIL_BUFFER_BIT)
```

Enabling

```
glEnable(GL_STENCIL_TEST)
```

Configuring

Setting the pass/fail criteria:

```
glStencilFunc(func, ref, mask)
```

The `func` argument specifies the test between `ref` value and the stencil buffer value for the pixel in question. It can be:

```
GL_NEVER, GL_ALWAYS, GL_LESS, GL_GREATER,
GL_EQUAL, GL_LEQUAL, GL_GEQUAL
```

The `mask` is bitwise ANDed with `ref` and the stencil value to achieve multiple stencil planes

Defining how to modify the buffer on pass/fail:

```
glStencilOp(fail, zfail, zpass)
```

The `fail` argument specifies what to do if the stencil test fails. If it passes, then `zfail` defines the action if the depth test is failed. If the depth test also passes then `zpass` is used. These actions can be:

```
GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR
```

Example

Restrict drawing to where stencil is non zero value.

```
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_GREATER, 0, 0xffffffff);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

    //Draw some things

glDisable(GL_STENCIL_TEST);
```

Partial Code Fragment

```
glDisable(GL_DEPTH_TEST);
glColorMask(GL_FALSE,GL_FALSE,GL_FALSE,GL_FALSE);

glClearStencil(0)
glClear(GL_STENCIL_BUFFER_BIT)

glEnable(GL_STENCIL_TEST);

//Replace buffer with 1's wherever we draw

glStencilFunc(GL_ALWAYS, 1, 0xffffffff);
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);

drawMirrorSurface();

glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glEnable(GL_DEPTH_TEST);

//Now draw only where the 1's are

glStencilFunc(GL_EQUAL, 1, 0xffffffff);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

glPushMatrix();
    glScalef(1.0, -1.0, 1.0);
    setLightSourcePositions();
    drawWorld();
glPopMatrix();

glDisable(GL_STENCIL_TEST);

setLightSourcePositions();

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    drawMirrorSurface();
glDisable(GL_BLEND);

drawWorld();
```

Scene Antialiasing

- OpenGL does not automatically perform full scene antialiasing
- Can do this using ‘jittering’
- Redraw the entire scene several times using different fractional pixel offsets in x and y each time
- Blend all the images together into one
- Use the accumulation buffer to collect these images and blend them (usually higher precision than the color buffer)

Usage

```
glClearAccum(0.0, 0.0 , 0.0, 0.0);
glClear(GL_ACCUM_BUFFER_BIT);

glAccum(op,value);
```

The `op` defines the operation, `value` is used differently depending upon the operation. `op` can be one of `GL_ACCUM, GL_LOAD, GL_RETURN, GL_ADD, GL_MULT.`

`GL_LOAD` – Loads the buffer with `value` times RGBA values in the current color buffer.

`GL_ACCUM` – Same, but adds to the buffer, rather than replacing.

`GL_RETURN` – Copies the buffer times `value` into the current color buffer.

Jittering

- Adjust our projection matrix to shift the viewpoint by a fractional pixel in x and y
- Optimum jittering values have been computed depending upon the number of jitters
- See Chapter 10 for `accPerspective()` and `jN[]` (code linked on web page)

Pseudo Code

```
Clear the accum buffer

Loop i from 0 to N

    Clear the color and depth buffers

    accPerspective(FOV,ASPECT,NZ,FZ,
           jN[i].x, jN[i].y, 0, 0, 1 )

    Draw the world

    glAccum(GL_ACCUM, 1.0/N)

Endloop

glAccum (GL_RETURN,1.0);
```

***Don't forget to ask GLUT for your buffers:

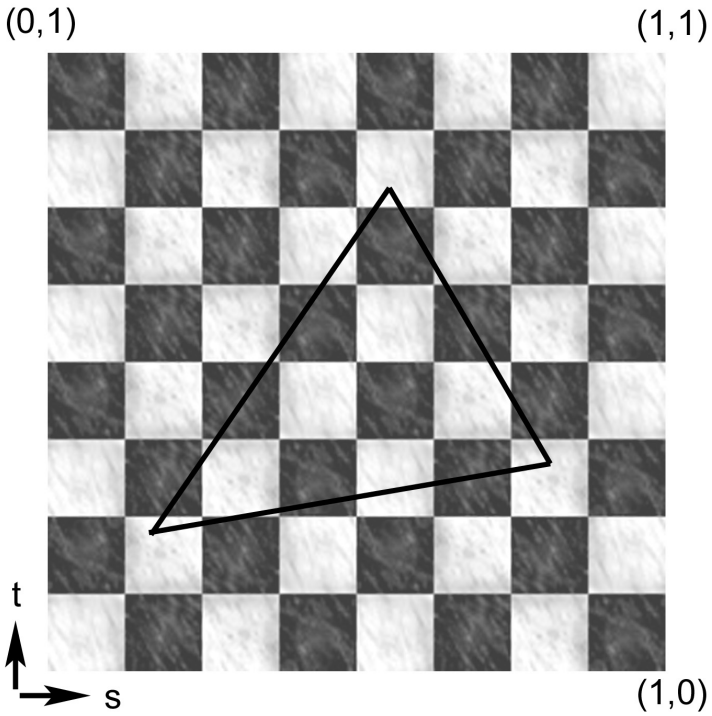
```
glutInitDisplayMode( bit1 | bit2 | … )
```

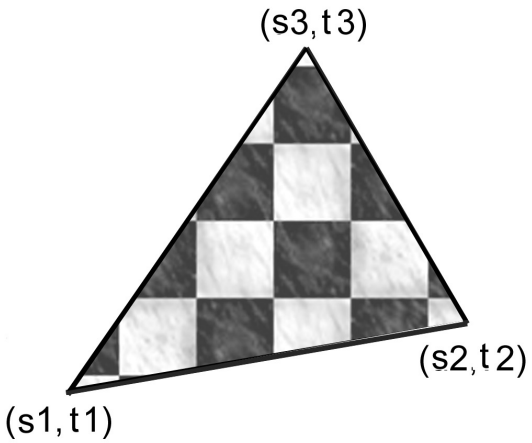
Bits:

```
GLUT_SINGLE, GLUT_DOUBLE, GLUT_RGB,
GLUT_ACCUM, GLUT_DEPTH, GLUT_STENCIL
```

Texture Mapping

- Create a texture object
- Indicate how the texture is applied to pixels
- Specify its texture data
- Enable texture mapping
- Draw the scene including geometric and texture coordinates





Creating a texture object

To requests a list of handles to n new texture objects:

```
glGenTextures(n,&texIndices[0])
```

Each one must be initialized. The following call initializes the new texture to the default states:

```
glBindTexture(GL_TEXTURE_2D,texIndex)
```

Subsequent texture state calls will affect, and be stored with, the currently bound texture.

Binding a texture already initialized will make it current, and its states can be edited.

Setting the Texture States

First define what happens if s or t fall outside of [0,1]. GL can repeat the texture, or clamp to the border color:

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_S, GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Setting the Texture States (cont'd)

Next we define how to filter the texture when the texture pixels are larger than screen pixels:

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, magFilter);
```

`magFilter` can be `GL_NEAREST, GL_LINEAR`

Finally we define how to filter the texture when the texture pixels are smaller than screen pixels:

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, minFilter);
```

`magFilter` can be `GL_NEAREST, GL_LINEAR,`
`GL_NEAREST_MIPMAP_NEAREST,`
`GL_NEAREST_MIPMAP_LINEAR,`
`GL_LINEAR_MIPMAP_NEAREST,`
`GL_LINEAR_MIPMAP_LINEAR`

Storing the texture image

To actually specify the texture image, we call to GLU to load an image array into our texture object, and simultaneously create mipmaps.

```
gluBuild2DMipmaps(GL_TEXTURE_2D,GL_RGB,
          width, height, GL_RGB,
          GL_UNSIGNED_BYTE, arrayPtr);
```

Setting the Texturing Function

Call to the following to decide how to apply the texture image to polygon surfaces (global):

```
glTexEnvf(GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE, mode)
```

Where `mode` is one of `GL_DECAL, GL_REPLACE, GL_MODULATE, GL_BLEND.`

The mode, and the texture's internal format (e.g. GL_RGB) decide how the texture is applied.

To get this looking good with lighting, use RBG textures, with GL_MODULATE to texture polygons drawn with white diffuse material properties.

Drawing

At draw time: enable texturing, set the texture function, bind the desired texture, and specify texture coordinates with your vertices.

Example Code Fragment

```
// Read texture into an RGB uint array (texData)

glGenTextures(1,&texIndex);
glBindTexture(GL_TEXTURE_2D, texIndex);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                                                GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                                      GL_LINEAR_MIPMAP_LINEAR);

glPixelStorei(GL_UNPACK_ALIGNMENT,1);
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height,
          GL_RGB, GL_UNSIGNED_BYTE, texData);

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)

glEnable(GL_TEXTURE_2D);

glBindTexture(GL_TEXTURE_2D,texIndex);  //Redundant, this is
                                // already currently bound

// Set to a white, diffuse material

glBegin(GL_TRIANGLES);
     glTexCoord2fv(&t[0][0]);
     glNormal3fv(&n[0][0]);
     glVertex3fv(&v[0][0]);

     glTexCoord2fv(&t[1][0]);
     glNormal3fv(&n[1][0]);
     glVertex3fv(&v[1][0]);

     glTexCoord2fv(&t[2][0]);
     glNormal3fv(&n[2][0]);
     glVertex3fv(&v[2][0]);

glEnd();
glDisable(GL_TEXTURE_2D);
```