Today's Agenda

> Check your github repos

> Overview of rendering pipeline

> Texturing

> Image Loaders

> GLU shapes (with texture coords)

> meshes / OBJ loaders

Rendering Pipeline

practically...

1. calculate PROJECTION matrix (the lens of your camera)
2. calculate MODELVIEW matrix (positions the drawing "cursor")
3. set-up LIGHTING
4. set-up MATERIALS
5. bind TEXTURES
6. send information to the graphics card :
   for each vertex, you can send a color, a normal, and a texture coordinate

```
glTexCoord2f(u, v); //between 0f and 1f, used for TEXTURING
glNormal3f(x, y, z); //used for LIGHTING CALCULATIONS
glColor3f(r, g, b); //between 0f and 1f
glVertex3f(x, y, z);
```

Textures

Texturing allows you to place images on top of your geometry.

The general idea has the following steps:

A. loading and initializing the texture
1) load an image file (jpg, bmp, png, etc) into your program
2) generate space for this texture and give that space an ID
3) bind that texture to the GL context
4) set parameters for the texture
5) load the image into the texture

B. using the texture (within your display loop)
1) enable texturing
2) bind the pre-loaded texture to the GL context
3) begin drawing geometry
for each primitive:
    4) pass in texture coordinates
    5) pass in vertices
6) stop drawing geometry
7) disable texturing

Textures

This is a lot of steps... many platforms (JOGL, P5, OF) simplify the steps for you.

If you are using GLUT there are some good utility libraries that handle loading images for you (SOIL, DevIL, libpng, etc).

Once you've loaded the image into the texture, it's sent directly to the graphics card, and using it thereafter is extremely fast. So in general, textures are loaded in the initialization phase of your application, but then used within the display loop.

Without going into too many details, here are commands for loading your image into a texture:

Textures

```
//load an image...
img = ... (see NeHe site for an .bmp example, or libpng for .png, etc)

//create space for one texture, which can be referred to by the int texID
int texID;
glGenTextures(1, &texID);

//bind the (empty) texture to the GL context
glBindTexture(GL_TEXTURE_2D, texID);

//define what happens if we go beyond the bounds of our texture
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

//define the filtering -- what happens if the geometry is bigger/smaller than the img
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_FILTER, GL_LINEAR);

//load the image into the texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imgW, imgH, 0, GL_RGBA, GL_UNSIGNED_BYTE, img);
```

Image Loaders

There are various helper libraries for loading in images into a byte array, or directly into an OpenGL texture. A simple cross-platform that supports jpegs, pngs, etc, is called SOIL ("simple opengl image loader"):

http://www.lonesock.net/soil.html

```
> cd wherever_you_put_the_soil_directory/projects/makefile
> mkdir obj
```

open up the makefile and replace the line starting with "CXXFLAGS" so that it indicates the 64 bit architecture:

```
CXXFLAGS = -arch i386 -arch x86_64 -O2 -s -Wall
```

```
> make
> sudo make install
```

Now you can use the SOIL library for opengl like so:

```
> gcc -lSOIL -framework GLUT -framework OpenGL -framework Cocoa main.c -o main ; ./main
```

Image Loaders

Example, call this function after the GL context is created and before the main loop:

```
GLuint texID; //id for the texture

void loadTextures() {
  texID = SOIL_load_OGL_texture (
     "your_image_file.jpg",
     SOIL_LOAD_AUTO,
     SOIL_CREATE_NEW_ID,
     SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y
    );

  /* check for an error during the load process */
  if( 0 == texID ) {
    printf( "SOIL loading error: '%s'\n", SOIL_last_result() );
  }
}
```

Textures

And here are the commands for applying the texture to geometry:

```
//turn on texturing
glEnable(GL_TEXTURE_2D);

//bind the (empty) texture to the GL context
glBindTexture(GL_TEXTURE_2D, texID);

//draw the geometry, indicating the texture coordinates
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0);
glVertex3f(-1.0, -1.0, 0.0);
glTexCoord2f(1.0, 0.0);
glVertex3f(1.0, -1.0, 0.0);
glTexCoord2f(1.0, 1.0);
glVertex3f(1.0, 1.0, 0.0);
glTexCoord2f(0.0, 1.0);
glVertex3f(-1.0, 1.0, 0.0);
glEnd();

//turn off texturing
glDisable(GL_TEXTURE_2D);
```

Textures

Texture coordinates range from 0 to 1 in each direction.

That is, along the x-axis, 0f = 0% of the width, and 1f = 100% of the width, etc

If you go outside the bounds of the coordinates, it the texture will either clamp to the boundary, or repeat, depending on the parameter you've set when initially loading the image into the texture.

You don't necessarily need to include the full range of the texture, and you can potentially place multiple sub-images within a single image.

Textures are a common way in which to send data to the graphics card to be used by GLSL shader, which we'll explore in a few weeks.

GLU geometry

```
//GLU

GLUquadricObj qObj = gluNewQuadric(); //load up new quadric
gluQuadricDrawStyle(qObj, GLU_FILL); //fill in the facets on the surface
gluQuadricNormals(qObj, GLU_SMOOTH); //make nice normals across the facets
//if lighting is enabled then the normals are already present
gluQuadricTexture(qObj, true); //automatically create texture coords for the object
//if you a bind a texture, then the texture coords are already present

gluSphere(qObj, 2f, 64, 64); //make a sphere with radius 2 and high resolution

//check out the glu library for other primitives
```

Loading models from external programs

Instead of creating complicated models directly in OpenGL, you can use one of many existing 3D modeling programs, such as Maya3D, or Blender. Both of these programs (as well as lots of others) will let you save your model as an .OBJ file.

The simplest .OBJ files store your mesh as a set of vertices, normals, and texture coordinates.

More complicated ones will also have an associated .MTL file, which can also store material properties of different parts of your mesh.

There are lots of libraries out there which will load and display these OBJ files for you. Or you can write your own.