Lab-PM OpenGL

In this exercise we will learn the basics of how to use OpenGL, glu and glut to render 3D. Recall that OpenGL is a state machine where you enable or disable features and modify the current state. OpenGL uses stacks to store the current states - for instance projection matrices and flags. Documentation of OpenGL 1.3, glu1.3 and glut version 3 is available online.

OpenGL – Open Graphics Library. A software interface to graphics hardware.
GLU – The OpenGL Utility Library. Complementary routines to OpenGL. Mainly mipmapping, matrix manipulation, polygon tessellation, NURBS and error handling.
GLUT- The OpenGL Utility Toolkit. A utility toolkit for writing window system independent OpenGL programs. Tools for creating windows and menus etcetera.

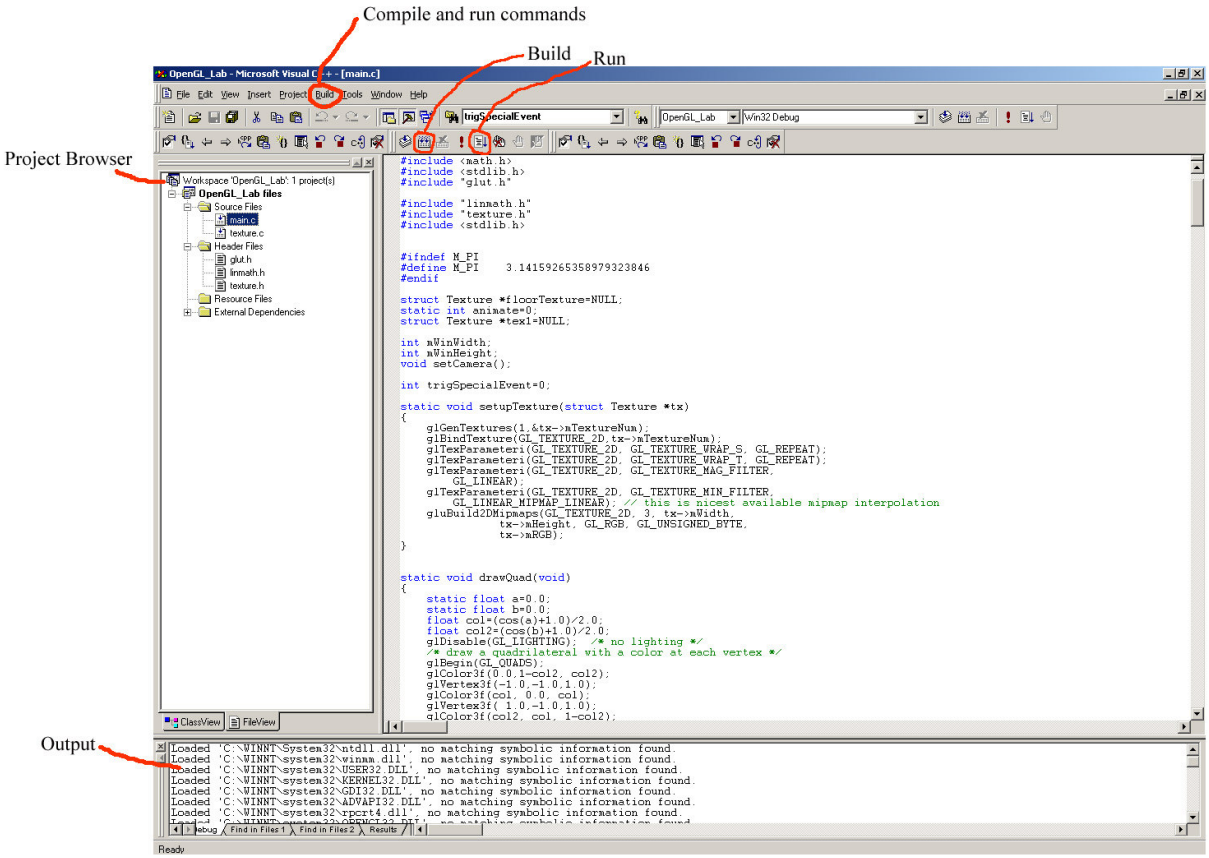
Documentation can be found online at:
http://www.ce.chalmers.se/staff/uffe/glspec13.pdf
http://www.ce.chalmers.se/staff/uffe/glu1.3.pdf
http://www.ce.chalmers.se/staff/uffe/glut-3.spec.pdf

Set-up

1. Download the lab from http://www.ce.chalmers.se/undergraduate/D/EDA425/labbar/lab1.zip
2. Extract the zip-file
3. Double-click on the file OpenGL_Lab.dsw to start Microsoft Visual C++. Note that the code is written in C, and not C++. This is controlled by the file extension (c vs. cpp).

A Quick introduction to Microsoft Visual C++

We will use Microsoft Developer Studio in these exercises. If you want to use another compiler, you may do so, but then we probably cannot provide any help regarding compiling and linking.



Project Browser: This browser shows all the files in the project. Double click a file to open it for editing. You can use the **Window-**menu to watch several files simultaneously.
Build Menu: Commands for building (compiling and linking) the project and running the program. There are also two toolbar buttons for this.
Output Window: This window shows the output of the compiler, linker, and etcetera. It does, however, not show the output of the program.
For Help on OpenGL commands: Use menu Help->Index. Type for instance glEnable as keyword.

PART 1 – Some OpenGL basics

1. Startup

Notice the structure of the program.
In **main** ():

- Set window states such as: size, double buffering, z-buffering
- Give glut the addresses to several callback-functions on events such as: mouse, keyboard, and redraw.
- Setup initial camera position

motion(): called on mouse movements
mouse(): called on mouse button events
handleKeys(): called for ordinary key events
handleSpecialKeys(): called for special keys like UP_ARROW, LEFT_ARROW etc.

Compile the project OpenGL_Lab (with button **F7**) and run the program (with button **F5**). You should see a large flat gray polygon. Navigate by pressing the left mouse button and move around.

2. Lighting

Study function **display**():

- First the color buffer and the z-buffer are cleared by calling glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
- Secondly, we setup the projection matrix and modelview matrix with a call to our function **setCamera**().
- Thirdly, we call our function **drawFloor**() that draws the gray polygon.
- Finally, we swap the front and back buffer with glutSwapBuffers(), which displays the frame, and then tell glut that we want to start a new frame with glutPostRedisplay();

Before **drawFloor**(), insert a call to setupLight(); which turns on lighting. Note that the material settings in **drawFloor**() will have effect now. Gouraud shading will be used because we set glShadeModel(GL_SMOOTH) in **main**(). Run the program and notice that the shading of the polygon varies smoothly over the surface. You can try GL_FLAT as well.

Add one more light source. OpenGL handles up to 8 light sources, which can be point lights, spotlights or directional lights (see help section for glLightfv)

3. Texturing

We are now going to add a texture to the floor.
First we need to read the texture to memory, which is stored in ppm-format. This is done with our **readPPM**() function in **main**(), since we only want to read it from disc once. Then the texture must be sent to OpenGL. This is done by **setupTexture**(), also called in **main** (). You can use the program convert.exe (needs magick.dll, X11.dll and Xext.dll) to convert images to .ppm format.

Study function **setupTexture**():

glGenTextures(1, &id); - generates one new texture ID number

glBindTexture(GL_TEXTURE_2D,id); - sets this number as active texture (remember that OpenGL is a state machine)

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

Indicates that the active texture should be repeated, instead of for instance clamped, for texture coordinates > 1.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
```

Sets the type of mipmap interpolation to be used on magnifying and minifying the active texture. These are the nicest, but often slowest, available options.

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, texture width, texture height, GL_RGB,
                  GL_UNSIGNED_BYTE, texture image data);
```

Sends the texture image to OpenGL and associates it with the texture id. It also creates the mip maps. This is a glu helper function instead of calling glTexImage2D() to specify each individual mip map.

When applying a texture to an object, texturing must be enabled with glEnable(GL_TEXTURE_2D); Which texture to use is defined by glBindTexture(GL_TEXTURE_2D, texture id);

In function **drawFloor**(), remove the comments on glEnable(GL_TEXTURE_2D), glBindTexture(GL_TEXTURE_2D, texture id) and glDisable(GL_TEXTURE_2D); Note that a texture coordinate is specified for each vertex, with glTexCoord2f(u,v). Run the program.

4. Geometry

OpenGL handles several primitive types, such as:

GL_POINTS
GL_LINES
GL_LINE_LOOP
GL_LINE_STRIP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
GL_POLYGON

We are now going to draw a triangle. Study the function **drawStuff**(). Between glBegin(GL_TRIANGLES); and glEnd() insert for instance the following triangle:

```
glNormal3f(0.4, 0.4, 0.8);
glVertex3f(1,1,0);
glVertex3f(-1,0,1);
glVertex3f(1,-1,1);
```

Run the program. You should see a gray triangle. Note that the normal usually should be normalized to give correct lighting. However, if glBegin(GL_NORMALIZE) is specified (we do this in setupLight), OpenGL does this for us automatically.

Set another material by adding the following lines above glBegin(GL_TRIANGLES) :

```
GLfloat red[4]={0.9,0,0,1};      // Note that these 2 lines must be at the beginning of the function in C
GLfloat darkred[4]={.1, 0, 0, 1};
glMaterialfv(GL_FRONT, GL_AMBIENT, darkred);
glMaterialfv(GL_FRONT, GL_DIFFUSE, red);
```

Run the program. The triangle should be red.
Add a texture with the following code:

Define globally at the top of file main.c similar to floorTexture:

```
struct Texture *tex1=NULL;     // Define globally at the top of file main.c similar to floorTexture
```

At bottom of **main**(), before glutMainLoop(), insert the following calls:

```
tex1=readPPM("white-marble.ppm");
setupTexture(tex1);
```

In **drawStuff**(), insert the following 2 lines before the call to glBegin(GL_TRIANGLES)

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, tex1->mTextureNum);
```

We must also specify texture coordinates for each vertex: Add 3 calls glTexCoord2f(0,0); glTexCoord2f(0,1);glTexCoord2f(1,1);. The code should look like this:

```
glBegin(GL_TRIANGLES);
        glNormal3f(0.4, 0.4, 0.8);
        glTexCoord2f(0,0);
        glVertex3f(1,1,0);
        glTexCoord2f(0,1);
        glVertex3f(-1,0,1);
        glTexCoord2f(1,1);
        glVertex3f(1,-1,1);
glEnd();
```

Run the program. Experiment with other material parameters. Instead of (0,1) and (1,1) as texture coordinates you could try other settings, like (0,3), (3,3).

5. Transparency

The most common way to create transparent objects is by using OpenGL's blending function. Add following lines in function drawStuff() somewhere before the call to glBegin(GL_TRIANGLES);. The first line enables blending. The second specifies how the blending should be done. With these parameters the destination will receive an interpolated color value of α*source color + (1- α)*destination color.

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

You must also set the alpha value in the current colors to something less than 1. For instance 0.5 (GLfloat red[4]={0.9,0,0,0.5};).
Note that to get correct transparent behavior, all overlapping transparent objects/polygons should be rendered in back-to-front order. It is common to first render all non-transparent objects in any order, and then sort all transparent objects/polygons and render them last. This transparency calculation does not require the presence of alpha bitplanes in the framebuffer.

6. Reflections with environment mapping

To fake reflections, environment mapping can be used. Add following lines in function drawStuff() somewhere before the call to glBegin(GL_TRIANGLES);.

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

Run the program. The textured triangle will probably now look like it reflects clouds in a sky. Remove the transparency to see the effect more clearly. Note that if we did not store the current OpenGL enable states with glPushAttrib(GL_ENABLE_BIT); we would have to call glDisable(GL_TEXTURE_GEN_S); and glDisable(GL_TEXTURE_GEN_T); to switch off environment mapping before exiting the drawStuff() function.

PART 2 – Animations and Events

1. Animating objects

1. Create a function that draws a teapot. Call the function from within display() so that the teapot gets redrawn each frame. For some reason the teapot is defined with clockwise ordering for front facing polygons, which differs from the more usual counterclockwise ordering for front facing polygons (Quake also uses clockwise ordering). To get facing right, we can use glFrontFace(GL_CW) to instruct OpenGL to use clockwise ordering instead.

```
glFrontFace(GL_CW); // set clockwise ordering
glutSolidTeapot(1);
glFrontFace(GL_CCW); // reset to counterclockwise ordering
```

2. Set a different material of the teapot.

3. Animate the teapot by affecting the modelview matrix. Preferably, you should save the old modelview matrix with glPushMatrix() and glPopMatrix(), and at least save the current state of enabled features with glPushAttrib(GL_ENABLE_BIT) and glPopAttrib(). Use glTranslatef(x,y,z), glRotatef(angle, axis_x, axis_y, axis_z), and glScalef(sizex, sizey, sizez). glRotatef(angle, axis_x, axis_y, axis_z) takes an angle in radians and the axis to rotate around.

(Hint: To make the teapot look like it bounces you could use a sinus function like: float yoffset = 0.5*sin(((float) (time%45)/45.0)*3.14); and call glTranslatef(0,yoffset,0), where time is incremented once each frame.)

4. We are now going to use glut's input callbacks to control the start and stop of the animations. Add a global Boolean variable that indicates whether or not animation is 'on' or 'off'. Toggle the Boolean in our function **handleKeys**(). **handleKeys**() is set in **main**() as the callback function for keyboard events with the call to glutKeyboardFunc(handleKeys). The variable should be toggled when a key of your choice is pressed. Modify the code so that animation is done only when the Boolean is true.

If you have time: add a texture to the teapot and animate the texture. Animate the color of the teapot.

2. Animating camera

Study function **setCamera**(). Animate the camera by affecting viewpos, viewat, and/or viewup each frame when animation is on.

3. Mouse Events

Study function **mouse**(). This function is set as the callback function when mouse buttons are pressed or released. The callback is set in **main**() with the glutMouseFunc(mouse)-command. Also take a look at the **motion**() function. This function is called every time the mouse is moved. The callback is set in **main**() with glutMotionFunc (motion).

Now, make something happen each time one of the mouse buttons are pressed. You could for instance make an object appear or make the teapot spin around.

Good Luck!

APPENDIX A: GOODS

Some useful things about OpenGL:

glPushAttrib(GL_ENABLE_BIT);
Pushes the current state of enabled and disabled functions onto the stack. Often used to save current state before calling a function that might affect the state. Note that there are several more bits that can be stored than just GL_ENABLE_BIT. See help on glPushAttrib().

glPopAttrib();
Pops the state.

glMatrixMode(GL_MODELVIEW);
Sets current matrix mode to the model view matrix. All following calls to glPushMatrix(), glLoadMatrix, glTranslate(), glRotate(), glScale and so on will affect the current model view matrix. Other arguments are GL_PROJECTION and GL_TEXTURE.

glPushMatrix();
Pushes the current matrix onto the stack.

glPopMatrix();
Pops the current matrix of the stack.