CS 185 Final Project

Assignment Description:

Our final project will build on the current project and allow improved control of viewpoint and rendering. In this lab the viewer/player will start in the middle of a room with objects from the 3D model library at various places in the room. The viewer should be able to look up, down, left right, and move through the room using simple commands. The viewer should not be able to go through (penetrate) the walls of the room.

Requirements:

1. Render a room with 4 walls, ceiling and floor.
2. Place several objects in the room.
3. Objects should have appropriate normals and back face removal.
4. Be able to move around the room using keyboard keys (arrows)
5. Be able to change the direction you are looking with the mouse (up, down, left, right)
6. Render objects using shaders you implement yourself including at least:
   a. Phong illumination (per-pixel rendering)
   b. Texture mapping of room walls
   c. Another advanced rendering technique of your choice, e.g. cartoon rendering, cube mapping, or bump mapping or something you come up based on your own research

Notes:

1. Use the camera position to see if you are behind walls (vector from camera to wall center vs. wall normal).
2. Use the provided initialization function as a starting point for loading shaders.
3. To access the new OpenGL functions to load and compile shaders on the Mac you have to include <OpenGL/glext.h>
4. On windows, you have to use the OpenGL extension wrangler in order to get pointers to all the functions you are going to use. The wrangler can be found at:
   a. http://glew.sourceforge.net/
   b. Instructions (also online):

```
// Download GLEW from http://glew.sourceforge.net/install.html
// build in release mode or get pre-built windows binaries.

// copy glew\glew-1.5.3\bin\glew32.dll to your windows\system32 directory

// Put the GLEW library and header files forwhere the compiler can see them,
// such as the visual studio include and library directories:

// copy glew\glew-1.5.3\lib\glew32.lib         (VC_ROOT)/lib
// copy glew\glew-1.5.3\include\GL\glew.h      {VC ROOT}/Include/GL
// copy glew\glew-1.5.3\include\GL\wglew.h     {VC ROOT}/Include/GL

// My copy of visual studio 8 has these directories (VC_ROOT) here:
// C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\Include\gl
///C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib

// In your program include <GL/glew.h> before <gl/gl.h> or <gl/glut.h>
// call the glew initialization function right after window creation:
//    glutCreateWindow (argv[0]);
//    GLenum err = glewInit();
//    if (GLEW_OK != err)
//          return 0;
//    }
// If you do not include the glew library, you will get link errors (opngl
//   functions not found)
// If you do not call glewInit() you will get runtime errors (NULL pointers)



// Function for loading shaders.  This function assumes the frgment
// and vertex shaders have same root name, e.g. tex.vert and tex.frag.
void printShaderInfoLog(GLuint obj)
{
      int infologLength = 0;
      int charsWritten  = 0;
      char *infoLog;

      glGetShaderiv(obj, GL_INFO_LOG_LENGTH,&infologLength);

      if (infologLength > 0)
      {
            infoLog = (char *)malloc(infologLength);
            glGetShaderInfoLog(obj, infologLength, &charsWritten, infoLog);
            fprintf(stderr, "%s\n",infoLog);
            free(infoLog);
      }
}

void printProgramInfoLog(GLuint obj)
{
      int infologLength = 0;
      int charsWritten  = 0;
      char *infoLog;

      glGetProgramiv(obj, GL_INFO_LOG_LENGTH,&infologLength);
```

```
	if (infologLength > 0)
	{
		infoLog = (char *)malloc(infologLength);
		glGetProgramInfoLog(obj, infologLength, &charsWritten, infoLog);
		fprintf(stderr,"%s\n",infoLog);
		free(infoLog);
	}
}

GLuint createShader(std::string file_name)
{
	GLuint vertex_shader = glCreateShader(GL_VERTEX_SHADER);
	GLuint pixel_shader = glCreateShader(GL_FRAGMENT_SHADER);

	// obtain file sizes:
	size_t fsize1, fsize2;
	GLint bytes_read;
	std::string vert_shader = file_name + ".vert";
	std::string frag_shader = file_name + ".frag";

	struct stat statBuf;
	stat(vert_shader.c_str(), &statBuf);
	fsize1 = statBuf.st_size;
	stat(frag_shader.c_str(), &statBuf);
	fsize2 = statBuf.st_size;

	FILE *f1 = fopen(vert_shader.c_str(), "r");
	FILE *f2 = fopen(frag_shader.c_str(), "r");
	if (!f1 || !f2)
		return -1;

	// allocate memory to contain the whole file:
	GLchar * shader_buffer[1];
	shader_buffer[0] = (GLchar*)malloc(std::max(fsize1, fsize2));

	// Read in the fragment shader
	bytes_read = (GLint)fread((void*)shader_buffer[0],1,fsize2,f2);
	glShaderSource(pixel_shader, 1, (const GLchar**)shader_buffer,
&bytes_read);

	// Read in the vertex shader
	bytes_read = (GLint)fread((void*)shader_buffer[0],1,fsize1,f1);
	glShaderSource(vertex_shader, 1, (const GLchar**)shader_buffer,
&bytes_read);

	free(shader_buffer[0]);

	GLint status;
	glCompileShader(vertex_shader);
	glGetShaderiv(vertex_shader, GL_COMPILE_STATUS, &status);
	if (status == GL_FALSE)
	{
		printShaderInfoLog(vertex_shader);
		return -1;
	}
	glCompileShader(pixel_shader);
```

```
	glGetShaderiv(pixel_shader, GL_COMPILE_STATUS, &status);
	if (status == GL_FALSE)
	{
		printShaderInfoLog(pixel_shader);
		return -1;
	}

	GLuint shader_program = glCreateProgram();
	glAttachShader(shader_program,vertex_shader);
	glAttachShader(shader_program,pixel_shader);

	glLinkProgram(shader_program);
	glGetProgramiv(shader_program, GL_LINK_STATUS, &status);
	if (status == GL_FALSE)
	{
		printProgramInfoLog(shader_program);
	}

	return shader_program;
}

void init()
{
	// Load two shaders with names default.vert, default.frag
	// and toon.vert, toon.frag.
	shader_program1 = createShader("../default");
	shader_program2 = createShader("../toon1");

	// Set up any uniform variables or texture maps again:
	if (shader_program3 != 0)
	{
		GLint loc = glGetUniformLocation(shader_program3, "diffuse_tex");
		glUniform1i(loc, 0);
	}

	//… any other initiaization
}

// and an easy way to update shaders without stopping the program:
void keyboard (unsigned char key, int x, int y)
{
	switch (key) {
		// other keyboard stuff

		case 'b': // for build
		{
			if (shader_program1 != 0)
				glDeleteShader(shader_program1);
			if (shader_program2 != 0)
				glDeleteShader(shader_program2);


			shader_program1 = createShader("../../default");
			shader_program2 = createShader("../../toon1");

			// Set up any uniform variables or texture maps again:
			if (shader_program3 != 0)
```

```
                  {
                        GLint loc = glGetUniformLocation(shader_program3,
"diffuse_tex");
                        glUniform1i(loc, 0);
                  }
            }
            default:
                  break;
      }
      glutPostRedisplay();
}
```

Extra credit ideas:

1. Add multiple light sources to the room
2. Add additional rendering effects, such as reflection, light maps, shadows, or something of your own choosing.
3. Move smoothly through the room. Turn along an arc instead of pivoting
4. Add additional rooms and the ability to visit them
5. Incorporate ability to shoot objects from last lab, and show visually where the hit occurs