15-462 Project 1: Basic OpenGL

Release Date: Tuesday, January 20, 2009

Due Date: Tuesday, February 3, 2009 at 23:59:59

1 Overview

In this project, you will have an opportunity to implement the basic OpenGL concepts described in the first few lectures. You will need to implement basic camera and lighting functionality, rendering of two primitive object types (spheres and triangles), and rendering of a water surface (represented as a triangle mesh).

Starter Code: `http://www.cs.cmu.edu/~15462/proj/01/p1.tar.gz`

Project 1 Lecture: `http://www.cs.cmu.edu/~15462/lec/03/lec03b.pdf`

2 Submission Process

Your handin directory may be found at
`/afs/cs.cmu.edu/academic/class/15462-s09-users/`*andrewid*`/p1/`.
You should submit the entire contents of the handout, your modified files, and any screenshots in this directory.

Regardless of what machine you use to code your assignment, it **must** compile and run correctly on the machines in the Wean 5336 cluster. There are a lot of students in the course and only 25 machines, so you probably don't want to wait until the last minute to test it.

In addition, you must fill out the `p1.txt` file in the handout, describing which features you have implemented for project 1, and which you have not. In particular, if you have missing features, be sure to explain how much you accomplished and why, and if you did extra work, be sure to explain what that was and how you did it. Furthermore, you should detail anything you think we need to know to understand your code.

3 Required Tasks

- Correctly set the projection and modelview matrices based on camera position.

- Create a heightmap for the watersurface in `WaterSurface::update`.
- Create a triangle mesh from the heightmap in `WaterSurface::update`.
- Create per-vertex normals for the triangle mesh in `WaterSurface::update`.
- Use GL lighting to add lights to the scene.
- Set material properties of the geometries so that they correctly shaded.
- Render the `WaterSurface`, `Sphere`, and `Triangle` objects, using the modelview matrix to correctly transform the objects.
- Your code should render the two staff scenes (0 and 1) the same as the reference images (included in the handout).
- You must submit some number of screenshots (with the names `shot`*nn*`.png`) along with your code. The screenshots must be taken from different angles than the default camera views. Do not submit an overly large number of screenshots.

4 Files to Edit

You will likely need to modify the following files:

- `project.cpp`: The main update/render functions
- `geom/watersurface.cpp/.h`: The `WaterSurface` class
- `geom/triangle.cpp/.h`: The `Triangle` class
- `geom/sphere.cpp/.h`: The `Sphere` class

You should definitely have a look at:

- `scene.h`: Scene-related classes
- `vec/vec.h`: Vector classes
- `vec/quat.h`: Quaternion class
- `vec/mat.h`: Matrix classes

You are free to modify any files unless they are specifically marked otherwise.

5 Camera Transforms

It is recommended that you begin by implementing camera transforms and lighting. We have provided `Camera`, `Light`, and `Scene` classes (all defined in `scene.h` which store the information that you will need (i.e. the camera position, a list of the scene's lights, etc.). Study these classes to learn how the camera and light data is stored within a `Scene` object, and what the accessor functions are. The starter code handles moving the camera around with the mouse and keyboard, so all you have to do is correctly render based on the given position.

You will need to modify the function `void prj_render(Scene* scene)` in `project.cpp`. Use the accessor functions to retrieve the camera's position, direction, and up vector, then use the OpenGL functions described in lecture to set camera projection using the `GL_PROJECTION` matrix and `gluPerspective`. Also, you will need to implement the camera transform by modifying the `GL_MODELVIEW` matrix usng `gluLookAt`.

6 Lighting

The scene contains a list of lights. You will need to add each light to the scene by enabling lights using OpenGL. For project 1, the staff scenes only use a single light. You must also use `glLight` to set the position and color of each light correctly.

7 Primitive Objects

In `void prj_render(Scene* scene)`, you should iterate over all geometries in the scene and invoke the `void Geometry::draw() const` function. You must implement this function for each geometry type using OpenGL calls. We suggest you first implement this function for the `Sphere` and `Triangle` classes.

8 Transformations

You will also need to make use of basic transformations on objects. This includes being able to properly scale, rotate, and translate objects. You can look into `glTranslate`, `glRotate`, and `glScale` to acomplish this. Note that the order in which you apply these matters. In this project, you should apply the transformations in the order of scaling, rotating, then translating. Note that if you are using the GL functions, this actually means that you should call translate first, then rotate, then scale.

Regarding rotations, you may note that the orientation of our objects is stored as a quaternion. The `Quat::to_angle_axis` function will be extremely useful regarding `glRotate` (be careful about radian/degree conversion).

9 Materials

Each object has a Material object that describes its material properties, such as the ambient, duffuse, and phong specular colors. You should use the `glMaterial` functions to correctly shade objects based on these colors. For project 1, the only memebers of the `Material` class that you must use are `ambient`, `diffuse`, `phong`, and `shininess`. You should use these to set the `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, and `GL_SHININESS` properties, respectively.

10 Water Surface

Once you have completed the steps above, you should be able to see objects in both demo scenes and move the camera using your keyboard and mouse. Your final task will then be to implement rendering of the water surface included in the second demo scene. The `WaterSurface` class, in `geom/watersurface.h`, also has a draw function. Unlike the primitive object types, however, the water surface has a more complex shape and is animated to change shape over time. You therefore need to also implement `void Watersurface::update(real_t time)` to create the model at each timestep.

You will need to construct a heightmap of the water surface and create a triangle mesh from that heightmap in the manner described in lecture. We have provided a function `WaterSurface::get_height` which will return the y-coordinate (height) of the water surface for a given (x, z) position and time. In its local space, the water surface's x and z values range from -1 to 1, or the unit cube. Two members, `resx` and `resz`, define the number of vertices to create in each direction. After constructing a triangle mesh, you will also need to create normals for each vertex in the mesh.

11 Extra Credit

Create additional geometry or updatable geometry types and use them in your own scenes. Note that simply creating additional scenes does not actually count for extra credit, although you will have the opportunity to pit them against your peers in the visual contest for p1.