CMSC 23700 **Introduction to Computer Graphics** **Project 1**
Autumn 2015 **October 2, 2015**

Basic OpenGL Rendering

Due: Monday October 12 at 10pm

1 Summary

This project is the first part of a three-part project, in which you will implement a number of standard rendering techniques. The goals of this project are to get your feet wet with simple graphics programming and to give you some quick feedback on the course submission and grading policies.

The project involves implementing a simple viewer that reads a scene from the file system that consists of one or more objects and then rendering the objects using either wireframes or flat shading. Your viewer should include controls for moving the camera.

2 Description

This project has three parts: loading, rendering, and user interface. We discuss each of these in more detail below.

2.1 Loading

Upon startup, your program will load a scene to be displayed. A scene is represented by a directory containing a JSON file called `scene.json` and one or more OBJ files. The `scene.json` file contains a JSON object with two fields; a JSON object that specifies the initial camera position and a JSON array that specifies the objects in the scene (see Figure 1 for an example). The initial camera specification has five fields:

1. The `size` field specifies the initial window size (1024×768 in the example in Figure 1).
2. The `fov` field specifies the **horizontal** field of view in degrees (120° in the example).
3. The `pos` field is a point that specifies the initial position of the camera ($\langle 0, 3, -10 \rangle$ in the example).
4. The `look-at` field is a point that specifies the initial point at which the camera is looking ($\langle 0, 3, 0 \rangle$ in the example). Note that this value is not the camera's direction vector.
5. The `up` field is a vector that specifies the initial up direction of the camera ($\langle 0, 1, 0 \rangle$ in the example).

```
{
  "camera" : {
      "size" : { "wid" : 1024, "ht" : 768 },
      "fov" : 120,
      "pos" : { "x" : 0, "y" : 3, "z" : -10},
      "look-at" : { "x" : 0, "y" : 3, "z" : 0},
      "up" : { "x" : 0, "y" : 1, "z" : 0}
    },
  "objects" : [
      { "file" : "box.obj",
        "pos" : { "x" : 0, "y" : 0, "z" : 0},
        "color" : { "r" : 0, "b" : 1, "g" : 0}
      }
    ]
}
```

Figure 1: An example `scene.json` file

Each object is specified by three fields:

1. The `file` field specifies a file name relative to the scene directory (`box.obj` in the example in Figure 1).
2. The `pos` field specifies the world-space coordinates at which the object should be placed (at $\langle 0, 0, 0 \rangle$ in the example).
3. The `color` field specifies the color of the object as an RGB triple (the color blue in the example).

We will provide a class `Scene` (see `scene.hxx`) that takes care of the details of loading the scene description and object descriptions. Your code will use the scene object to initialize your view state and object representations. In other words, your code should load the camera description into your `View` object and it should initialize your OpenGL buffers from the objects descriptions.

2.2 Rendering

Your program should support two rendering modes. The first is a wire-frame mode, where just the edges of the objects's triangles are rendered. Look at the documentation for `glPolygonMode` to see how to do wireframe rendering. The second is a flat-shading mode, where the triangles are rendered in the object's color without any lighting calculations. To implement rendering, you will need to write a basic shader program (you can use the same one for both wireframe and flat-shading modes) and you will need to complete the implementation of the `WireframeRenderer` and `FlatShadingRenderer` classes.[1]

2.3 User interface

Your viewer must support the following keyboard commands:

[1] You may make modifications to their common base class (`Renderer`) if you wish.

`f F`	switch to flat-shading mode
`w W`	switch to wireframe mode
`q Q`	quit the viewer.

In addition, it should provide either keyboard or mouse-based camera controls that allow one to view the scene from any point.

You should feel free to play with different schemes, but here is one possible strategy. Define the horizontal plane to be the plane containing the camera perpendicular to the camera's **up** vector. Define the view axis to be the line through the **look-at** point that is perpendicular to the horizontal plane. Then we could use the left and right arrow keys to rotate the camera around the view axis. We can define another axis as the line in the horizontal plane through the **look-at** point that is perpendicular to the vector from the **look-at** point to the camera and then let the up and down arrow keys rotate the camera around it. Note that these rotations change the position and orientation of the camera, so the axes also change. Lastly, we use the plus and minus keys to control moving the camera towards and away from the **look-at** point. We would specify a minimum and maximum distance to the **look-at** point to keep things under control.

3 Sample code

To get you started, we will seed your repository with sample code. The code will be organized into a directory called `proj1` with four subdirectories:

- `build`, which contains a `Makefile` for compiling your project.
- `shaders`, which is where you should put your shader source files.
- `src`, which will hold the C++ source code for your project. Initially, this directory contains some scaffolding code to get you started.
- `scenes`, which contains sample scenes for you to test your program on.

4 Summary

For this project, you will have to do the following:

- Design a representation of objects that you can render in OpenGL and write code to convert the OBJ representations to your representation.
- Set up the initial camera state from the scene; you will also need to connect this state to your shader code.
- Write a simple shader that renders objects in solid color without lighting.
- Implement the rendering setup code for wireframe and flat-shading modes.
- Design and implement camera controls.

5 Submission

We have set up an **svn** repository for each student on the `phoenixforge.cs.uchicago.edu` server and we will populate your repository with the sample code. You should commit the final version of your project by 10:00pm on Monday October 12. Remember to make sure that your final version **compiles** before committing!

History

2015-10-05 Corrected UI description to match sample implementation.

2015-10-02 Original version.