15-462 Project 0: Basic OpenGL

Release Date: Thursday, August 27, 2009
Due Date: Tuesday, September 15, 2009

Starter Code: http://www.cs.cmu.edu/~15462/proj/p0.tar.gz

1 Overview

For this project, you will have the opportunity to familiarize yourself with basic OpenGL concepts described in the first few lectures. You will begin by implementing basic camera and lighting functionality and rendering a 3-D model.

We will only look at a simple triangle mesh model. You must render the model using simple lighting and materials with OpenGL. You may find the OpenGL Programming Guide (Red Book) to be extremely helpful, as it covers all topics needed for this assignment in great detail.

2 Submission Process and Handin Instructions

1. Your handin directory may be found at
   /afs/cs.cmu.edu/academic/class/15462-f09/handin/andrewid/p0/.
   All your files should be placed here. Please make sure you have a directory
   and are able to write to it well before the deadline. We are not responsible
   if you wait until 10 minutes before the deadline and run into trouble.

2. Do not add levels of indirection when submitting. For example, your
   makefile should be at .../andrewid/p0/Makefile, not
   .../andrewid/p0/myp0/Makefile or .../andrewid/p0/p0.tar.gz.
   Please use the same arrangement as the handout.

3. You should submit all files needed to build your project, as well as any
   models or screenshots that you used or created. Your deliverables include:
   
   • src/ folder with all .cpp and .hpp files.
   • Makefile
   • All *.mk files
   • p0.txt

   Be aware that you have a limit to your AFS space, so do not submit unreasonably large number of models or images.
4. Please do not include:
   - The bin/ folder or any .o or .d files.
   - Executable files

5. We will enter your handin directory and run make, and it should build correctly. The code must compile and run on the WeH 5336 cluster machines. Be sure to check to make sure you submit all files and it builds correctly.

Failure to follow submission instructions will negatively impact your grade.

3 Required Tasks

**Input:** The input to the program is a OBJ format model and a camera. The starter code loads the model into a structure of vertices and triangles, and sets the camera values based on mouse input.

**Output:** The output is a rendering of this model with basic lighting, in addition to screen shots and p0.txt. You must submit a few screen shots of your program and fill out p0.txt. Some example screenshots are included in the handout.

![Figure 1: Example program output](image)
For your program, you must:

- Correctly set the projection and modelview matrices based on the camera’s values. The user should be able to move the camera around with the mouse.
- Compute per-vertex normals for the triangle mesh of the input model.
- Use GL lighting to add lights to the scene to correctly shade the model with basic materials.
- Render the input model as a triangle mesh using points, lines, and fill modes.
- Submit a few screen shots of your program’s renderings.
- Use good code style and document well. We will read your code.
- Fill out `p0.txt` with details on your implementation.

At a minimum, you must modify `p0.cpp` and `p0.txt`, though you may modify or add additional source files.

`p0.txt` should contain a description of your implementation, along with any information about your submission of which the graders should be aware. Essentially, if you think the grader needs to know about it to understand your code, you should put it in this file. You should also note which source files you edited.

## 4 Starter Code

It is recommended that you begin by first reviewing the starter code is provided. Though the amount you must edit is small, we are providing you with a large code base to get you started. You will need to get familiar with it, since much of it will be used in future projects as well.

### 4.1 Programming Language

The code base is in C++. We know that this may be a new language to many of you, and so it is limited to C and a small subset of C++. In addition to C, the code uses classes, operator overloading, and a bit of the Standard Template Library, notably `std::string` and `std::vector`. If you are unfamiliar with these, your first task should be to learn them. The project uses C++ over straight C since we find classes and operator overloading extremely useful when implementing matrix algebra. Vectors, matrices, and quaternions are much more verbose and cumbersome to use in straight C than in C++.

Hopefully this introduction project will help you familiarize yourself with the language beyond what you know about C. It is small enough that even someone with no prior experience with C++ or OpenGL can finish the project well before the due date. This project does, however, assume strong familiarity with the C programming language, computer systems, and basic linear algebra.
4.2 Building the Code

The code is designed to run and build on the SCS Linux machines. These are currently located in WeH 5336 and the Habermann clusters. The code should also successfully build on any school linux machine.

4.3 What You Need to Implement and Provided Code

The code that you are required to implement is located in p0.cpp. There are 4 shell functions which you should modify to implement your project. You may additionally edit any other source files in the handout, though you must keep the basic program behavior the same. To add additional source files, edit the list in sources.mk.

We have taken care of creating a window and valid OpenGL context for you to draw with. Additionally, we have provided some basic functionality such as taking a screen shot via a right-click menu. We also have code that controls a camera via mouse left and middle clicks and the control and shift keys.

We have also provided several math classes for linear algebra which define vectors, quaternions, and matrices. You should look at the header files carefully, since we provide several math functions so that you will not have to implement them, and they can serve to make numerically intensive code much more readable.

5 Grading: Visual Output and Code Style

Your project will be graded both on the visual output (both screen shots and running the program) and on the code itself. We will read the code.

Therefore, part of your grade is dependent on your code style. This includes both how you structure your code and how readable it is. For this and future projects, you should think carefully about how to implement the solution in a clean and complete manner. A correct, well-organized, and well-thought-out solution is better than a correct one which is not.

Since we read the code, please remember that we must be able to understand what your code is doing, so you should write clearly and document well. If the grader cannot follow along with the code, then the grader will not be able to easily award partial credit for incorrect output, nor will the grader be able to provide useful feedback on the correctness of your code.

We will also be looking for correct usage of the C language, such as making sure memory is freed, avoiding using uninitialized memory, and many other common pitfalls. These can impact your grade. Additionally, we will provide comments on your C++-specific usage, though these will generally not affect your grade since it is a new language for many of you. More general style and C-specific style (i.e., rules that apply in both C and C++) will, however, affect your grade.
6 Implementation

6.1 Setting the projection and camera position

The first thing to do is set OpenGL’s transformation matrices to correctly view the scene. We include a class that contains all information needed to position a camera in a scene. This includes things such as position, direction, up vector, and field of view. To use this information in OpenGL, one must set the two main transformation matrices, the PROJECTION matrix and the MODELVIEW matrix. Consult the Red Book for further information on how to use and manipulate OpenGL matrices.

The projection matrix usually contains the camera’s field of view (FOV), aspect ratio, and near and far planes. You may find \texttt{gluPerspective} useful here. The modelview matrix contains the camera’s position, direction, up vector, and the model’s position, orientation, and scale. We only have to deal with the camera for now. For this, you may find \texttt{gluLookAt} useful.

6.2 Rendering the Model

The model is defined as a list of triangles and a list of vertices. A triangle is simply three indices of a vertex. Vertices are a position and texture coordinate. You can ignore the texture coordinates unless you plan to add textures to your model.

OpenGL is able to render many geometric primitives. The program provides OpenGL with the vertex data for each primitive sequentially. You will be using the \texttt{GL_TRIANGLES} primitive. You may use any OpenGL method to render your model. If you are new to OpenGL, the course staff recommends using vertex arrays. Essentially, you put all the primitive data into an array and draw the entire array with one function call. They are simple to use yet more efficient than more basic methods. See the Red Book section on vertex arrays for more details.

OpenGL can render polygons such as triangles in 3 ways:

1. Point, which draws single points at each vertex.
2. Line, which draws lines that run along the edges of the polygons.
3. Fill, which fills the entire triangle.

This is controlled with \texttt{glPolygonMode}. In your program, the mode is controlled via the right-click menu. One of the functions you must implement is called by the program to change the polygon mode. Your program must render all 3 by implementing this function.

The rasterized primitives are placed in the color buffer. Note that the color buffer is not cleared automatically each frame, and so you must do this manually. Look at \texttt{glClear} to accomplish this.

Note that we must use the depth buffer to make sure primitives are drawn in the correct order. Closer primitives should be drawn on top of farther primitives. We can use OpenGL’s depth test for this. You must enable the depth test using
glEnable to accomplish this. As with the color buffer, the depth buffer is not automatically cleared. This must be done with glClear.

6.3 Lighting the Scene

Once you render a model correctly with the transformation matrices set correctly, you should be able to see a solid silhouette of your model when rendered using the fill mode. The last task is to add some lighting to the scene to provide somewhat realistic looking shading.

The Red Book has entire chapters on lights and materials. We will only make you do the very basics for this assignment. Create a single light for your scene, and set some ambient, diffuse, and specular materials for your object in order to light it. You can also play with the color. The Red Book chapter on lighting contains information on how to do this.

6.4 Computing Model Normals

For lighting to work correctly, you will need normal data for your model. We intentionally do not load any normals from the model file; you must compute them yourself. We ask you to compute per-vertex normals, which can be done as follows:

1. Compute the normal for each triangle by taking the cross product of two edges and normalizing. Note that the order in which you take the cross product is important, as otherwise all the normals will be backwards. The default convention for OpenGL is that primitives are in counter-clockwise order. Be aware that some models you find may use clockwise rather than counter-clockwise, and you must somehow deal with that issue if this is the case. Be sure to normalize this vector as well.

2. Set the normal for a particular vertex to be the average of the normals of all triangles that it touches. Again, you should normalize this vector.

7 Words of Advice

• Start early. Though this assignment is simple, it may take you time to become familiar with this style of programming. There are a lot of concepts introduced.

• Familiarize yourself with all the OpenGL concepts (transformation matrices, primitive rendering, lighting, materials, etc) before starting the assignment. The Red Book covers all these topics, as do many sources on the internet.

• Familiarize yourself with the structures in math/ and scene/ before starting, as they will be used throughout the semester. In particular, have a look at the Vector, Camera, and Model structures.

• Start with a simple object, even a hard-coded one, at first.
• Make sure you have a submission directory that you can write to as soon as possible. Notify course staff if this is not the case.
• Any Linux computer on campus may be used to build the project. However, you must still make sure that it compiles on the 5336 cluster machines.
• Experiment! Play around with OpenGL, even features that aren’t part of the assignment. There is a lot you can do with OpenGL.