CS 4620 Program 1: Hello OpenGL

out: Tuesday 7 September 2010
due: Tuesday 21 September 2010

Authors: Prof. Doug James, Ivaylo Boyadzhiev (TA), Colin Ponce (TA)

In this first programming assignment, you will have fun learning how to build an interactive graphics application in OpenGL (`http://www.opengl.org`). There are several graphics functionality requirements that you must satisfy, however what you create, be it a video game or a strange virtual world, is totally up to you.

1 Getting Started

Your primary references for OpenGL are their website (`http://www.opengl.org`), and the "OpenGL Programming Guide – The Red Book" [3] for which v1.1 is available online (`http://www.opengl.org/documentation/red_book`). You will also be able to find other online references to learning OpenGL programming online, e.g., NeHe tutorials (`http://nehe.gamedev.net`), however all code in your assignment is expected to be your own. In addition, **your write-up must document all external references/sources that you use to complete your assignment.**

Some framework code is provided to get you started, and is available for download from CMS (`http://cms.csuglab.cornell.edu`). Both Java and C++ implementations are available depending on your coding preference and what you would like to learn more about. The starter code will setup a basic OpenGL application that renders a simple 3D scene in perspective, along with some other basic functionality. However, the framework does not contain any code that does anything particularly interesting—you will develop that code yourself. Questions about the starter code should be directed to Ivaylo Boyadzhiev (C++, `iboy@cs.cornell.edu`) and Colin Ponce (Java, `cponce@cs.cornell.edu`).

1.1 About the starter code

See Appendix A for more info about the C++ starter code, and Appendix B for more info about the Java starter code. The starter code comes with an example of how to draw (using display list), rotate and texture a 3D cube in perspective projection.

1.1.1 Moving around the world

Both C++ and Java starter codes come with implemented camera movement functionality.You can use your mouse to rotate camera view and use the arrow keys to go Front/Back/Left/Right, furthermore by pressing U/D you can go Up and Down.

1.1.2 Making a video

You can start/stop capturing frames by pressing the "R" key. Frames will be exported in `.PNG` format with the following file name pattern: `exporter{ID}-XXXXX.png`, where `ID` is a number that will be incremented each time you start new recording and `XXXXX` are ascending numbers of the corresponding frames. *Note that between different runs of your program, ID will be reset to 0, so you should save your favourite movies between start/stop of your application or they will be overwritten!*

You can combine your PNG files into an AVI file with different tools, here is an example of how to do that using ffmpeg (`http://ffmpeg.org`):

ffmpeg -qscale 6 -r 25 -b 9600 -i exporter0-%05d.png movie.avi

-qscale controls the quality, 1 (highest quality) to 31 (lowest quality)
-r defines the output FPS
-i exporter0-%05d.png takes as an input all frames of our first animation (ID is 0)

2 What To Do

While the over-arching goal is to produce an interesting graphics artifact, and have fun learning, there are several specific goals that your project should explore:

1. **Make a plan:** As a first step, you'll want to come up with an idea or vision for what you'd like to produce. If it's a simple game, think out the game play and come up with a simple design on paper, and think about what you need to implement in order for it to come together. If you're modeling a huge maple tree, think about the parts and what primitives you'll need, etc. As you decide what to do, you also need to figure out how to satisfy the other constraints of the project. You do not need to submit your design plans, however a good planning will show in your finish results.

2. **Rendering basics:** Since this is the beginning of the course, only simple shading and lighting is expected. Texture mapping is not required (although you can include it if you want). Perspective rendering is supported by the started code, although you can use orthographic projection if your application requires it.

3. **Drawing primitives:** You application is expected to draw objects using available 3 GL primitive types, e.g., `GL_POINT, GL_TRIANGLE, GL_QUAD`, etc. You can also use GLUT to draw the 6 platonic solids (incl. teapot ;).

4. **Transformations:** To spatially position/orient and animate objects in your scenes, you should make generous use of the available GL transformation commands, i.e., `glMatrixMode, glMultMatrix, glPushMatrix, glPopMatrix, glScale, glTranslate, glRotate.` Transformations are also useful for changing your viewpoint by suitably transforming the scene via the modelview matrix.

5. **Display Lists:** You will find that as you draw more and more objects, your application will run more slowly. One simple way to accelerate the repeated drawing of rigid objects where the same drawing, e.g., `glBegin/glEnd` commands, are executed, is to use pre-compiled display lists. Note that not all commands can be embedded in a display list. See the OpenGL "Red Book" for details.

6. **Instancing:** Using display lists, you can draw the same geometry repeatedly, i.e., instance it, while changing parameters such as transformations and colors outside the glBegin/glEnd drawing commands. Your application should use display lists to instance numerous objects. If you find that you are drawing many objects which are "off screen," you may wish to perform some form of *view frustum culling.*

7. **Interaction:** You should use keyboard and mouse input to interact with your program. Please document your user interface.

8. **Animation:** Static environments are not very interesting, so be sure to include interesting and relevant animation. You can use time-dependent transformations, or parametric geometry to produce simple animations.

9. **Textures** are important for most applications, and a simple example is included in the starter code. However, no particular texturing capabilities are expected or required in this assignment. We will explore more sophisticated lighting and texturing methods in later assignments, however you may use textures if you need them in your particular application.

10. **Text:** You may wish to rendering text to the screen. You can find information on printing text to the screen in the OpenGL "Red Book" (or Qt, GLUT, etc.). You will find that there are many possible ways to do this traditionally, each with it's own pros/cons [1].

11. **More advanced features** can be incorporated depending on what you are trying to do, e.g., collision detection and response, level-of-detail (LOD) control, view frustum culling (to avoid drawing off-screen objects), and textured rendering. You may wish to load external geometry, e.g., free models that you find on TurboSquid, to make your scenes more interesting—document your sources. Your professor and TAs are good sources of information on additional features.

12. **Make a movie:** Using the starter code, you will be able to export movies and produce *your video highlight* for the class to see. We might want to explore offline (noninteractive) rendering for very complex scenes.

3 Ideas for Projects

Try to devise a project that is interesting to you, can be described in a sentence ("what are you doing?"), and that can be implemented in a short project. Here are a few project ideas if you get stuck:

- **Make a video game** that's fun to play, visually interesting, that exercises your OpenGL muscles. Remake a classic game in a new way, e.g., 3D Tetris, or invent one of your own. Create a racing or flying game with simple graphics that is challenging and fun to play.

- **Model something** that is familiar and visually interesting, such as a tree or a building, or a familiar kid's toy. Model a meadow, and see how many blades of grass you can draw and animate at one time. Model an alien world with strange creatures that move around, and have interesting behaviors. Model an underwater world, with interesting creatures and float around with them. Model artificial life in an aquarium. Use falling particles to model Taughannock Falls.
- **Procedural (rule-based) modeling** can be an easy way to produce a lot of detail with a little bit of code. Examples include buildings, fractals (plants, mountains, etc), L-system grammars for plants, recursive structures for architectural models, etc. Try modeling a world entirely out of bricks (or LEGO) or spheres using OpenGL transformation commands. Try drawing as much stuff as you can and dump frames to disk to make a cool movie.
- **Other ideas:**
  - Model a fractal terrain and fly around it.
  - Explore the use of Perlin noise to model clouds, terrain, etc. [4, 2].
  - Build your own solar system. Try modeling planets using a random process.
  - Try making a gigantic randomized universe, and explore it.
  - Design some strange virtual plants that grow and cover your virtual world.
  - Model an ant colony with a million ants.
  - Model a futuristic city, such as "Machine City" from the movie "The Matrix Revolutions," using procedural geometric primitives. Compile it to display lists and explore your creation.
- Model a 3D interface for an operating system, or make an awe-inspiring 3D screen saver.

4 Submission and FAQ

A FAQ page will be kept on the course website detailing any new questions and their answers brought to the attention of the course staff.

Working in Groups: You can work by yourself or with one other person. If you work in a group, it is expected that your assignment will in some way do a little more than if you just worked on your own.

Submission Checklist: Submit your assignment as a zip file via CMS. You need to submit:

- **Documented code:** Include all of your code and libraries, with usage instructions. Your code should be reasonably documented and be readable/understandable by the TAs. If the TAs are not able to run and use your program, they will be unable to grade it. Try to avoid using obscure packages or OS-dependent libraries, especially for C++ implementations.
- **Brief report:** Include a description of what you've attempt, special features you've implemented, and any instructions on how to run/use your program. In compliance with Cornell's Code of Academic Integrity, please include references to any external sources or discussions that were used to achieve your results.

- **Video highlight!** Include a video that highlights what you've achieved. The video footage should be in the default starter-code resolution, and be no longer than 10 seconds. We will concatenate all of the class videos together to provide a summary of your work.
- **Any additional results:** Please include additional information, pictures, videos, that you believe help the TAs understand what you've achived.

Evaluation: Your work will be evaluated on how well you demonstrate proficiency with the requested OpenGL functionality, and the quality of the submitted code and documentation, but also largely on how interesting and/or creative your overall project is.

Appendices

A C++ starter code

We have set up a Qt4 (`http://qt.nokia.com`) based starter code for the C++ enthusiasts. We have chosen Qt (version 4.6) because it gives you a reach set of ready to use functionalities like loading/storing images, using timers and various graphics widgets including OpenGL drawables.

You can get help on Qt/OpenGL from the following web sites:
`http://doc.qt.nokia.com/4.6/examples-opengl.html`
`http://doc.qt.nokia.com/4.6/index.html`
Qt integrates very well with eclipse, here you can see how to do that:
`http://qt.nokia.com/developer/eclipse-integration/`

A.1 Setting up you environment

A.1.1 Linux

Download Qt4 for Linux from `http://qt.nokia.com/downloads` or using the software package management system for your distribution.

1. `unzip cpp_opengl_skel.zip -d cpp_opengl_skel`
2. `cd cpp_opengl_skel`
3. `qmake -project "QT += opengl"`
4. `qmake`
5. `make [debug|release]`
6. `./cpp_opengl_skel`

If you have more than one version of Qt installed, you might have to specify which version of Qt you want to use. For example if you have Qt3 as well, qmake might correspond to this version and you will have to use qmake-qt4 instead. Also it is important to write exactly "QT += opengl" as Qt is case sensitive.

A.1.2 Windows

Download and install Qt4 for Windows (`http://qt.nokia.com/downloads`). Add the following directories to your PATH environment variable: `[path_to_qt]\bin`; `[path_to_qt]\mingw\bin`; `[path_to_qt]\qt\bin`;

1. `Unzip cpp_opengl_skel.zip to cpp_opengl_skel`
2. `Open console windows (Windows->Run->cmd)`
3. `cd cpp_opengl_skel`
4. `qmake -project "QT += opengl"`
5. `qmake`
6. `mingw32-make [debug|release]`
7. `copy [debug|release]\cpp_opengl_skel .`
8. `cpp_opengl_skel`

Note that in step 7 we copied the executable in the main directory of the project, this is important, because the application will look for texture files in the current directory, so `box_texture.bmp` should be located inside the directory from which you start the program. Also it is important to write exactly “QT += opengl” as Qt is case sensitive.

A.2 Writing your code

The files that will be of main interest to you are glwidget.{h,cpp}, they contain the declaration and implementation of our glDrawable object.

Basic How-To:

- **OpenGL initialization** You can call various OpenGL initialization routines from `void GLWidget::initializeGL()`.
- **Draw your objects** You should call all your drawing routines from `void GLWidget::paintGL()`.
- **Animate your objects** `void GLWidget::animate()` will be called 25 times per second (this can be configured). You can put your animation logic here to create frame-based animations. Note that during capturing your animation might seem sluggish. This is because we are using frame-based animation instead of a time-based one. You can get a smoother animation after that when you combine your captured frames into a video file.
- **Texture your objects** There is an example code in `void GLWidget::loadTextures()` that loads a texture.

B Java Starter Code

We have set up starter code in Java using JOGL (Java OpenGL). JOGL is a wrapper library for the C++ implementation of OpenGL. That is, except where necessary, JOGL doesn't do anything except pass your function calls on to the C++ code.

JOGL is currently maintained by JogAmp (`http://jogamp.org`). It is highly recommended that you read the tutorial at the following web page, especially sections 2 and 3:

`https://sites.google.com/site/justinscsstuff/jogl-tutorials`

Note that this tutorial assumes you are using Eclipse, but sections 2 and 3 work just as well if you aren't. You can find the Javadoc on the current version of JOGL at

`http://jogamp.org/deployment/jogl-next/javadoc_public/`

B.1 Setting up your environment

We assume that the JRE and JDK are already installed on your system. To install JOGL, go to the following web page

`http://jogamp.org/deployment/autobuilds/`

and select the latest build of jogl (NOT jocl). At the time of of this writing that will take you to

`http://jogamp.org/deployment/autobuilds/jogl-b163-2010-08-30_15-36-01/build/`

From there download the version of JOGL that is appropriate to your system. Then unzip the file, and place the following files in your Java installed extensions folders:

1. `gluegen-rt.jar`
2. `jogl.all.jar`
3. `nativewindow.all.jar`
4. `newt.all.jar`

Your installed extensions folder depends on your operating system. If you don't know where it is, check out

`http://download.oracle.com/javase/tutorial/ext/basics/install.html`

Note that you must place these in both the extension folder for your JDK as well as for your JRE. It won't compile without one, and it won't run without the other. If you have multiple versions of the JDK or JRE on your system, make sure you place the files in the correct folder, or just place it in all of them.

Now go back to the .zip file you downloaded and take al of the .dll files and copy them to the `/bin/` folders of the Java JDK and JRE. For reference, on Windows those files are

1. `gluegen-rt.dll`
2. `jogl_desktop.dll`

3. `jogl_es1.dll`
4. `jogl_es2.dll`
5. `jogl_gl2es12.dll`
6. `nativewindow_awt.dll`
7. `newt.dll`

We also use the vecmath library, which is part of Java3D. Download this at

`http://www.oracle.com/technetwork/java/javase/releases-136574.html`

Be sure that, after the install, `vecmath.jar` is in the proper extension folders like the above .jar files. If it is not, you may have to copy the file in from the folders to which it is installed.

B.2 Writing your code

Use the Javadoc mentioned above to know what exactly to import into each of your files. You will likely need to import at least

1. `javax.media.opengl.*;`
2. `javax.media.opengl.glu.gl2.*;`
3. `javax.media.opengl.awt.*;`
4. `com.jogamp.opengl.util.*;`

Note that this is done for you already in the starter code.

Read the tutorial mentioned above to learn how the code is structured in JOGL. The basic started code is implemented in `GraphicsApp.java`. You will need to edit the `init()` function to alter what OpenGL initializes. The `display()` function gets called on repeat during animation, but it calls the `update()` and `render()` functions, and those do the real work. The `update` function updates the condition of the world that OpenGL is drawing; any simulation or animation code should happen here, and will likely involved little to no actual OpenGL code. The `render()` function tells OpenGL what to draw and how to draw. This function will likely be almost entirely OpenGL code. These are not the only functions you will have to edit in `GraphicsApp.java`, they are only the basic structural elements.

The file `JOGLEventListener.java` contains mouse, key, and action event listeners. You only need to alter this code if you are changing how the user interacts with the world.

References

[1] Steve Baker, *Fast Text in OpenGL*, `http://www.sjbaker.org/steve/omniv/opengl_text.html`.

[2] Paul Bourke, *Perlin noise and turbulence*, `http://local.wasp.uwa.edu.au/~pbourke/texture_colour/perlin`.

[3] J. Neider, T. Davis, and M. Woo, *OpenGL. Programming guide*, Addison-Wesley Reading, MA, 1997, `http://www.opengl.org/documentation/red_book`.

[4] K. Perlin, *An image synthesizer*, ACM SIGGRAPH Computer Graphics **19** (1985), no. 3, 296.