COMS W4160— Computer Graphics Spring 2013

Programming Assignment 1

out: Feb 7, Thursday
due: Feb 21, Thursday before the class

In this 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. While implementing these requirements, you are encouraged to use your imagination and creativity. Your program can be a video game, a strange virtual world or a clip of OpenGL animation. It is totally up to you.

1 Getting Started

Numerous reference materials and tutorials exist online (such as the OpenGL Website, the OpenGL red book, the NeHe tutorials, and the supplemental materials we referred in class and coursewiki). You are allowed to use any publicly available references to assist your programming. However, all code in your submission is expected to be your own. In addition, **your submission must include a writeup documenting a brief description of your implementation and all external references/sources used to complete your assignment**.

A C++ starter code can be found on coursewiki. It illustrates the way to setup a basic OpenGL application that displays a simple 3D scene in perspective, along with some other basic functionality. However, it contains no code that does anything particularly interesting—you need to develop that code yourself.

1.1 About the starter code

See Appendix A for details of compiling the C++ starter code. The starter code comes with an example of how to draw a teapot using glut. We also provide a screen capture to save the current frame buffer into a picture file. You can use it to dump frames and make a video in your submission (see section 1.3).

1.2 User interaction

We provide the code to process user interactions in the file trackball.h/.cpp. You are welcome to modify the code for any functionality needed in your code. If you feel the trackball implementation is not particularly suitable for you purpose, you can implement your own user-interaction code.

1.3 Making a video

In the starter code, you can capture the current frame by pressing "r" key. It generates a .ppm image file in the current directory. To automatically capture time series of frames, you can call the capture() method in the code when you finish drawing a frame. Those time series of screen shots are then used to generate a video. There are a few ways to make video. Firstly, it is suggested to convert the .ppm image files into .png files. This can be easily done by imagemagick software. On Linux, you can easily install it by

```
sudo apt-get install imagemagick
```

for Ubuntu system, and by

```
sudo yum install ImageMagick
```

for Fedora/Redhat system. Once the software is installed, you can simply run

```
convert screenshot-1.ppm screenshot-1.png
```

for the conversion. For a sequence of image files, run the following command

```
for((i=1;i<=N;++i)); do convert screenshot-${i}.ppm screenshot-${i}.png ; done
```

where N is the total number of frames. If you have commercial software such as Adobe Photoshop installed, you can also perform the conversion using its batch processing mode.

The next step is to combine your PNG files into a video. Again, there are different tools for this purpose. Here are few examples. If you installed quicktime pro, you can open a sequence of image files in quicktime and save them into a video. Another option is to use free software such as ffmpeg:

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

or

```
ffmpeg -r 25 -i screenshot-%05d.png -vcodec png movie.avi
```

where -qscale controls the quality, 1 (highest quality) to 31 (lowest quality), and -r defines the FPS.

2 Programming Requirements

While the key goal is to produce an interesting graphics artifact along with fun learning, there are several specific goals that your code should satisfy:

1. **Make a plan:** As a rst step, youll want to come up with an idea or vision for what youd like to produce. If its 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 youre modeling a huge maple tree, think about the parts and what primitives youll need, etc. As you decide what to do, you also need to gure 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 nish 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 as shown in the starter code).

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 nd 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 checkout the trackball.h/.cpp files, and 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, parametric geometry to produce simple animations or animations produced by user interactions.

9. **Textures:** 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** is optional for this assignment. You may wish to rendering text to the screen. You can find information on printing text to the screen in the OpenGL "Red Book". There are many possible ways to do this online, each with its own pros/cons.

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. This will give you bonus points for the assignment.

12. **Make a movie:** Using the starter code, you will be able to export screen shots and produce your video highlight for the class to see. As an option, you are allowed to explore offline (noninteractive) rendering for very complex scenes.

3 Ideas for Projects

Before staring to code, you should have a clear idea about what you plan to do. Your plan needs to be practical and implementable in a short project. While we highly encourage creativity and imagination, here are a few ideas if you get stuck:

- **A video game:** thats 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 kids toy. Model a meadow in central park, 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.

- **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.
- **A 3D screen saver for an operating system:** Generate stylized motion of light flows in space. Generate an abstract artistic dynamic motion effects.
- **Other ideas:**
  - show an animation of the growth of a tree.
  - Model a fractal terrain and y around it.
  - 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 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
  - Illustrate how a complex mechanics structure (e.g. a car engine) works.
  - Design your Rube Goldberg machine and show how it goes.

4 Submission and FAQ

Submission Checklist: Submit your assignment as a zip file via courseworks. Your submission must consist of the following parts:

1. **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. To ensure the TAs can grade your assignment, it is highly suggested to compile your code on CLIC machines, and include details of compiling and running your code on CLIC.

   In the starter code, we also include a CMakeLists.txt file which is used to automatically detect libraries and generate Makefile using the cmake building system. It is optional to modify the CMake file for your project, while it will probably make the compile more organized and less tedious.

2. **Brief report:** Include a description of what youve attempt, special features youve implemented, and any instructions on how to run/use your program. In compliance with Columbia's Code of Academic Integrity, please include references to any external sources or discussions that were used to achieve your results.

3. **Video highlight!:** Include a video that highlights what youve achieved. The video footage should be in a resolution of 960×540, and be no longer than 10 seconds. We will concatenate some of the class videos together to highlight some of your work.

4. **Additional results (optional):** Please include additional information, pictures, videos, that you believe help the TAs understand what you have achieved.

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.

A C++ starter code

We have set up a glut-based starter code. On Linux or Mac OS, you can compile the starter code by the following command

```
unzip pa1_starter.zip && cd pa1_starter && make
```

As you add your implementation files, you need to modify the Makefile to compile your code. To make it easier, we also include the configuration file for using cmake, a cross-platform building tool, to build the code. On Ubuntu, you can install cmake by the command

```
sudo apt-get install cmake
```

The following command sequence will make code compiled on both Linux and MacOS.

1. `unzip pa1_starter.zip`
2. `cd pa1_starter && mkdir gcc-build && cd gcc-build`
3. `cmake ..`
4. `make`

Library dependence: The starter code depends only on the OpenGL library and GLUT. Both of them have been installed on CLIC machines. Most of the operation systems with graphics interface should have them already. If not, they can be manually installed. On Ubuntu Linux, you can install them by

```
sudo apt-get install libglu1-mesa-dev freeglut3-dev
```