out: Feb 7, Sunday
due: Feb 21 at 10pm (submit to courseworks)

In this programming assignment, you will learn how to build an interactive graphics application in OpenGL. This assignment consists of two parts. In the first part, you need to implement geometric transformations and several graphics functionalities using OpenGL. In the second part, you are encouraged to use your imagination and creativity. Your program can be a video game, creation of a virtual world, or a clip of OpenGL animation.

1 Getting Started

Plenty of reference materials and tutorials exist online (such as the OpenGL Website, the OpenGL red book, the LWJGL Gitbook, and the supplemental materials we referred in class and course website). 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 your implementation ideas, important details, and all external references/sources used to complete your assignment**.

A Java starter code can be found on courseworks. It illustrates the way to setup a basic OpenGL application that displays a simple 3D scene in perspective, along with some other basic functionalities for user interaction using the Java OpenGL library LWJGL. Yet, it contains no code that does anything particularly interesting—you need to develop that code yourself.

Important: If you are using eclipse, you should add -XstartOnFirstThread on your VM argument

Grading. You code will be graded using **Java version 8** and **LWJGL 3**. In addition, the starter code also use the Java math library **JOML** and a library, **PNGJ**, to save a screen capture into a PNG file, both of which are included in the starter code package (under the folder `lib`). Please make sure your code can be successfully compiled under these settings. External Java libraries are allowed, but you need to include them in your submission as `.jar` files.

Please do not use other's code or download code online. When grading, we are going to run plagiarism detector over all submissions.

1.1 About the starter code

We have set up a Java starter code using LWJGL 3. The starter code comes with an example of displaying a cube with flat shading (see Figure 1). We also provide a screen capture to save the current frame buffer into a picture file. The functionality is provided by the method `Renderer.writePNG(Window window)`. Please note that if you are using a retina display, you might need to change the code of `writePNG` a bit. Please refer to the code comments to see the potentially needed changes. You can use it to dump frames and make a video in your submission (see section 3.1).

You can compile the source code in your favorite IDE (such as Eclipse). Or, if you installed the Apache Ant and if you are using Linux or MacOSX, after downloading the starter code you simply open a terminal

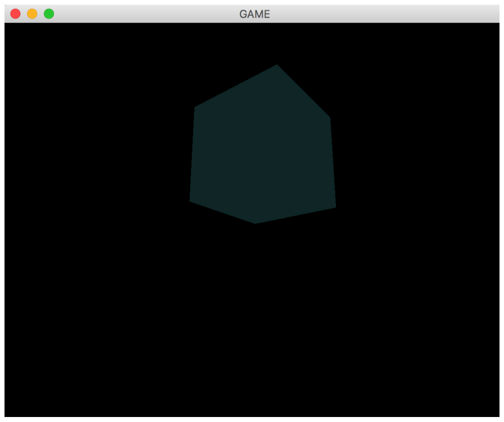

Figure 1: Uninteresting display of the starter code.

and type:

```
unzip PA1.zip && cd PA1 && ant
```

To successfully run ant, you need to specify the library path of LWJGL in the file build.xml. Please refer to the comments in build.xml to see how to edit the file.

2 Part 1: Programming Requirements

While the key goals are to produce interesting graphics and have fun, there are several other specific goals that your code should satisfy:

Loading a triangle mesh. Your starting point is to load a triangle mesh file stored in .obj file format. The OBJ file format is a simple data-format that represents 3D geometry alone. You can refer to the online format specification for its format details. Your task is to complete the method of OBJLoader.loadMesh(String filename) which returns an instance of the Mesh class. (You need to read the code and understand the data structure of the Mesh class first.)

After your implementation, please uncomment Line 64 of the file DummyGame.java to enable the use of OBJLoader.

Geometric transformation. After you complete OBJLoader, its method loadMesh returns an instance of the class Mesh. In this class, there are several empty methods that you need to implement in order to transform the triangle mesh.

1. translate(Vector3fc tran) Translate the mesh by a given vector.
2. scale(float sx, float sy, float sz) Scale the mesh by sx, sy, and sz along the x-, y-, and z- direction, respectively.
3. rotate(Vector3fc axis, float angle) Rotate the mesh around the given axis by the degree angle. The rotation angle is given in degrees.

4. reflect(Vector3fc p, Vector3fc n) Reflect the mesh with respect to the given plane. The plane is specified by the equation
$$(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{p}}) \cdot \vec{\boldsymbol{n}} = 0.$$

After successfully implementing these methods, you should be able to press "9"

Camera projection. Lastly, you need to implement the projection, view, and model matrices yourself **without using the OpenGL calls**. The projection, view, and model matrices are transformation matrices used to project 3D objects on a 2D screen. And their details will be talked in the lecture (at Feb. 9). The methods you need to complete are in Transformation.java.

3 Part 2: Creative Scene

After finishing the required functionalities, you'll have the freedom to create your own demonstration, whether a piece of animation, a game, or some special effects. Before starting 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.

3.1 Making a video

In the starter code, you can capture the current frame by pressing “1” key. It generates a snapshot.png file in the current directory. To automatically capture time series of frames, you can call the snapshot() method in the code when you finish drawing a frame. Those time series of screen shots are then used to generate a video by concatenating the pixel images together. There are different tools for this purpose. Here are a 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%d.png movie.avi
```

or

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

where -qscale controls the quality, 1 (highest quality) to 31 (lowest quality), and -r defines the FPS, and screenshot-%05d.png take all the PNG files with a format like screenshot100.png.

If you are using Mac OS or Windows, there are many screen capture tools to record a video directly. Those tools include FRAPS, Copernicus, and a list of tools on Wikipedia. You are allowed to use any of these tools to make a video.

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. To ensure the TAs can grade your assignment, it is highly suggested to compile your code with Java 8, and include details of compiling and running your code.

   In the starter code, we also include a build.xml file which is used to ease your compile and run the code using the Apache Ant building system. It is optional to modify the ant building file for your project, while it will probably make the compile more organized and less tedious. It will also ease the grading work of TAs.

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 the best 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.