CS 4053/5053
Homework 01 – Building Basic Graphics Applications with JOGL
Due Tuesday 2020.02.04 at 11:00pm.

All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment you will learn how to develop basic graphics applications using OpenGL, Java, and Gradle. Specifically, you will learn how to write code to animate simple sets of points and lines using the JOGL API, organize the code into a single-frame application in Java, and compile and run the application using Gradle.

Learning about Gradle

All homework assignments will involve writing in Java and building and running using Gradle. Visit https://docs.gradle.org/current/userguide/userguide.html to get a sense of what Gradle does and how it works. You **don't** have to install Gradle on your system. You don't even have to do any Gradle scripting. I've done that for you.

In the `homework01` download, go into the `ou-cs-cg` directory in `Build`. Read the `about.txt` file to learn about running Gradle tasks on the command line and the tasks you're most likely to need. Feel free to take a peek at the `build.gradle` script file. The *Alternative Start Scripts* section at the end describes how to create scripts to run additional main() classes. Most likely you won't need to create any, but the option exists. Don't otherwise change `build.gradle`!

Using Gradle with Eclipse

Gradle and Eclipse play well together. First, make sure you're using a version of Eclipse with the BuildShip plugin installed. Most recent Eclipse for Java distributions come with it. Second, run `gradlew eclipse` in the `ou-cs-cg` directory to prepare it for import into Eclipse. Third, import the `ou-cs-cg` directory as a *Gradle / Existing Gradle Project* using the Eclipse import wizard. Eclipse will add a *Gradle Tasks* pane to the editing window for running Gradle commands. (Other IDEs are less well supported, but can still be used to edit code.)

Building and Running Applications

To compile the build, type `gradlew installDist` on the command line. You need to be in the `ou-cs-cg` directory for this to work. Then go into the `build/install/base/bin` directory and run any of the resulting programs by typing its name. Each program has a `.bat` version for running on Windows. (Note that the `gradlew run` shortcut launches only the `base` program.)

The `base` program is very...well, basic! Its purpose is to help you learn one way of organizing JOGL code for simple graphics applications. Read through the code in the corresponding class, `Base.java` . You can find the source code deep in the `src` directory. The package structure is organized to hold code both for your homework assignments and for the example applications that we'll see in class. (The source code for the various examples will be revealed progressively; the download for each homework assignment will include an updated `ou-cs-cg` directory with the additional code and a suitably modified `build.gradle`.)

Exploring an Example

Study how a simple point-drawing program can be written using JOGL. The `lorenz` program displays an animation of the Lorenz attractor (https://en.wikipedia.org/wiki/Lorenz_system). (For differential equations like in the Lorenz system, one could integrate to calculate points precisely, such as with a high-order Runge-Kutta method. However, simple iteration and addition works for our purposes. See http://www.algosome.com/articles/lorenz-attractor-programming-code.html.)

The corresponding code is in the `Lorenz.java` class. Study the code and how it's organized to learn more about how to update and render an animated scene with JOGL. In particular, look at how the number of points to draw, `m`, increases exponentially but resets each time it hits a cap. Then look at how those points are iteratively calculated and drawn as a set using `GL_POINTS`.

Creating a New Example

It's finally time to *create* some animated point and line art of your own using OpenGL. Start by reading about the Tinkerbell map (https://en.wikipedia.org/wiki/Tinkerbell_map). Then write code to animate an increasing number of points, like in the Lorenz example. For the number of points, set the cap and rate of increase to produce an interesting and pleasant visual result. Choose the four Tinkerbell map constants likewise. Adjust the point locations to show the whole scene; there are several ways to do this. When the number of points resets after passing the cap, switch to line segments using `GL_LINE_STRIP`. Switch back and forth following each subsequent reset.

For this assignment, navigate to the `edu.ou.cs.cg.assignment.homework01` package and modify `Application.java` to implement your Tinkerbell example. Don't create any new Java files or directories for subpackages. (You'll be allowed to create both in future assignments.) You may add private methods if it helps to modularize your code. Document your code appropriately. The corresponding program to run is `hw01` in the `build/install/base/bin` directory.

Turning It In

Turn in a complete, cleaned, renamed, zipped copy of your `homework01` directory:

- Take a screenshot of your application window when it's in an interesting graphical state.
- Put the screenshot in the `Results` directory as `snapshot.png` or `snapshot.jpg`.
- Go into the `ou-cs-cg` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.
  - Run `gradlew clean` to reduce the size of your build.
  - If you're using Eclipse, run `gradlew cleanEclipse` and delete the `bin` directory.
- Append your 4x4 to the `homework01` directory; mine would be `homework01-weav8417`.
- Zip your entire renamed `homework01` directory.
- Submit your zip file to the Homework01 assignment in Canvas.

These steps will make your submissions smaller and neater, which speeds up grading a lot.

You will be scored on how accurately you have reproduced a recognizable Tinkerbell map, how well the animation works as described above, and...how cool it looks!