**CS 4053/5053**
**Homework 03 – Interaction and Paths**
*Due Tuesday 2023.03.07 at 11:00pm.*

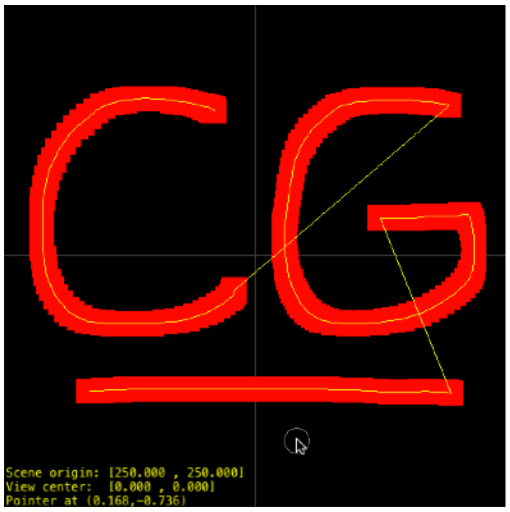
*All homework assignments are individual efforts, and must be completed entirely on your own.*

In this assignment you will learn how to add interaction to a moderately complicated 2-D scene created using JOGL. Specifically, you will use the mouse and keyboard interaction capabilities of Java to let the user manipulate objects, curves, and paths in a scene rendered in OpenGL.

## ***Getting Started***

Specifying curves (of shapes) and paths (of animation) are two of the most common activities in computer graphics. Interactive specification of curves is common in drawing applications. It is also used in game development to design more complex scene features and movement paths.

Consider the screenshot on the right. It shows an example of a hand-drawn curve made by accumulating mouse drag locations within a basic scene. Run the `interaction` program to try it out yourself. Study the `edu.ou.cs.cg.application.interaction` code to see how the program integrates interaction handling.



```
Scene origin: [250.000 , 250.000]
View center:  [0.000 , 0.000]
Pointer at (0.168,-0.736)
```

Actual rendering of shapes and animation involves various calculations on interactively specified points. One particularly important kind of calculation is the generation of curve segments from a few control points. Another is concatenation of segments to form longer curves. In animation, segments are usually decimated to show progression over time, both within segments and from segment to segment along the curve. All of the curves and paths used in this assignment can be generated using the basic parametric functions discussed in class (review the slides for details).

## ***The Implementation Process***

To get started, go to the `edu.ou.cs.cg.assignment.homework03` package in `ou-cs-cg`. It's a copy of the `interaction` package. First, integrate the pipeline, rendering, and animation code from your homework02 `Application.java` into the equivalent sections of `View.java`. (If you prefer, **you may integrate my code** from `edu.ou.cs.cg.assignment.solution02` **instead**.) Second, plan out how you want your scene to incorporate interaction, in accordance with the specification below. Third, add an appropriately named+typed variable to `Model.java` to represent each of the user-adjustable parameters and data in your scene. Fourth, access the model whenever the scene generation code needs a variable value. Fifth, add event handling to adjust the values of those variables in `KeyHandler.java` and `MouseHandler.java`. Finally, test the interactions individually, and in combination, to make sure it all functions as you intend!

You may add Java files (and subpackages) to the `edu.ou.cs.cg.assignment.homework03` package. You won't need to. Regardless, all new code must be yours. Organize the code inside each class as you like, but keep readability in mind. Avoid long methods, group related methods into sections, and document your code thoroughly. Remove any unused code left over from the `interaction` package before you turn it in. The corresponding program to run is `hw03`.

## ***Creating an Interactive Scene***

Imagine enhancements to the scene from your homework02 (or my solution02), then write code to view and interact with it. Feel free to be creative with your scene enhancements, but be sure to include all of the following features:

- Clicking and dragging the mouse *in the sky* creates colored points to augment the galaxy. Pressing the `<q>` or `<w>` key deletes the oldest or newest point, respectively (if there is one).
- Using the keyboard selects the sky color at the horizon, cycling through five allowed colors.
- Using arrow keys moves a feature around your scene...in a sensibly limited manner.
- Using number keys (regular *or* numpad) adjusts the geometry of a house feature, such as the number of sides on a star, the number of segments in a roof, or the number of window panes.
- Clicking the mouse in one of the fence boards flips its direction. *Color that board differently from the others to show that it is the flippable one.*

Also include any two of the following in your scene:

- Clicking in your novel house feature slides it to somewhere else inside the house, following a cubic path. Clicking it again moves it back to the original point along the same path.
- Clicking three times *in the sky* repositions the kite (instantaneously) and draws the string as a cubic path down to its anchor point. Simulate wind by cycling the kite along a closed curve.
- Clicking on the street creates a beachball that bounces down the street, less high as it goes.
- Clicking the flag moves it up, then down, its pole. Slow down the animation closer to the ends by using a cubic curve to calculate linear position along the pole in non-linear time (“easing”).
- Clicking the hydrant causes five new drops to shoot out in random arcs, pooling in the street.
- Clicking in the moon sets its shadow angle and animates it for one lunar cycle (= 5 seconds).
- Clicking in the sky adds a star. Shift-clicking in a star removes it. All stars move slowly along parallel quadratic paths from beneath the horizon to above the scene, repeating endlessly.

All of the items in the second list involve animation somehow. You may preserve or remove your animations from homework02 as you like, so long as they don’t conflict with the new animations.

It may be necessary to differentiate mouse and keyboard inputs to support some combinations of features. If that happens, use the Shift and/or Control modifier keys to disambiguate inputs.

Precise testing for clicks in/on shapes requires hit detection, which we haven’t covered yet. For now, simply define rectangles that roughly cover your click targets, and use their x and y limits to test for point containment. If your rectangles overlap, use modifier keys to pick which to test.

## ***Turning It In***

Turn in a complete, cleaned, renamed, zipped **COPY** of your **ENTIRE** `homework03` directory:

- Never delete `About` or `Results`! Preserve all file structure and contents of the assignment's original zip download, except for any modifications and additions specified in the instructions.
- 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`.
- Briefly list all interaction features, which inputs control them, and how they affect the scene.
- Put the list in the `Results` directory as `features.txt`.

- Go into the `Build/ou-cs-cg` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.
  - Run `gradlew clean` (on the command line). This should remove the `build` directory, reducing the size of your submission. We will clean and rebuild when we grade regardless.
  - If you used an IDE, remove any IDE-specific leftovers (such as the Eclipse `bin` directory).
- Append your 4x4 to the `homework03` directory; mine would be `homework03-weav8417`.
- Zip your entire renamed `homework03-xxxx####` directory.
- Submit your zip file to the Homework03 assignment in Canvas.

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

You will be scored on: (1) how many of the specified interaction and animation requirements that you satisfy; (2) how well your chosen features work individually and collectively as a part of the scene; (3) the overall appeal of your new features, especially animation paths; and (4) the clarity and appropriateness of your code.