CS 4053/5053
Homework 04 – Vectors and Reflection
Due Tuesday 2023.03.28 at 11:00pm.

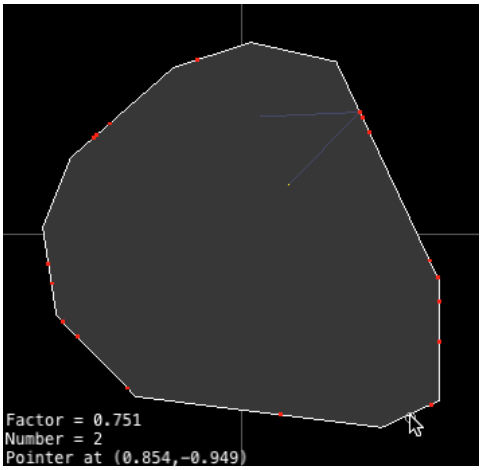
All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment, you will learn how to animate graphics with basic collision dynamics using vector calculations and the JOGL OpenGL graphics library. Specifically, you will write code to calculate and draw the ricocheting path of a moving point inside a convex polygon.

Getting Started

Before you start, review the slides on vectors, geometric objects, line intersection, and reflecting trajectories. I also recommend that you review chapter 4 of the textbook before starting. As you work on each part below, focus on how to model objects and movements as points and vectors.

Consider the screenshot on the right. It renders a small yellow point bouncing around inside an irregular convex polygon. Its recent trajectory is traced in light blue, and its recent hits on polygon sides are marked with red squares.



The Implementation Process

In this assignment, you'll start with a redacted version of the code that I wrote to create the screenshot. Go to the `edu.ou.cs.cg.assignment.homework04` package in `ou-cs-cg`. It's a copy of my `solution04` package (*which of course I'm not providing yet!*) but with key parts removed from the `View` and `KeyHandler` classes. Inside those two classes are sections commented as `TODO`...for you to do!

You shouldn't need to modify the other three files in the `homework04` package. If you like, you may add Java files (and subpackages) to the package, but you shouldn't need to. Regardless, all new code must be yours. Structure your code clearly and document it thoroughly, particularly your implementation of the reflection algorithm. The corresponding program to run is `hw04`.

Implementing Reflection with A Little Interaction

I recommend that you work in the following order on the parts commented with `TODO`:

- `View.drawObject()` — draw a single point
- `View.drawCursor()` — draw a cursor (*your design*) around the current mouse location
- `View.createPolygon()` — populate the collection with the vertices of a regular polygon
- `View.edgePolygon()` — draw a polygon's sides
- `View.fillPolygon()` — fill a polygon's interior
- `KeyHandler.keyPressed()` — map key presses into changes to parameters in `Model`
- `View.dot()` — calculate the dot product of two vectors
- `View.isLeft()` — test whether a point is to the left of a line defined by two points
- `View.contains()` — test whether a point is strictly inside a polygon
- `View.updatePointWithReflection()` — implement the reflection algorithm

Your implementation of the reflection algorithm will need to maintain state between updates. Add members to represent the reference vector, tracer points, and bounce points. Put the members under the two `TODO`s in `View` at the end of the Private Members section. Initialize the members as needed under the corresponding `TODO`s in the `View` constructor.

The reflection implementation will take the bulk of your time and effort. It's tight coding. Take it step by step, make sure you understand what each step does, and carefully check edge cases.

Drawing the Movement Tracer and Bounce Point

Once your reflection implementation is working, add members to store and code to draw the moving object's trajectory (in the polygon) and hit locations (on the polygon) over the previous 1.00 seconds of animation. To do that, complete the following parts commented with `TODO`:

- `View.updatePointWithReflection()` — add any new trajectory and/or bounce points
- `View.update()` — remove any old (> 1 second) trajectory and/or bounce points
- `View.drawTracing()` — draw the trajectory (*your choice of style*) in the polygon
- `View.drawBounces()` — draw the hit locations (*your choice of style*) on the polygon
- `View.clearAllTrace()` — remove all trajectory and bounce points

Whenever the shape of the polygon changes, the location of the moving object might fall outside of the new polygon. `Model` moves the object to the center to keep that from happening. It also calls `View.clearAllTrace()` to reset recording of trajectory and bounce points. Note that `Model` doesn't reset the object's *direction* or *speed*. If your code is correct, this shouldn't matter!

Turning It In

Turn in a complete, cleaned, renamed, zipped **COPY** of your **ENTIRE** `homework04` 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`.
- 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 `homework04` directory; mine would be `homework04-weav8417`.
- Zip your entire renamed `homework04-xxxx####` directory.
- Submit your zip file to the Homework04 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 completely and correctly you realize reflecting point movement and interactive adjustment in the listed parts; (2) the clarity and appropriateness of your code; and (3) the clarity and completeness of your comments, especially for the reflection algorithm.