CGRA151 2019 Assignment 3: splines and polygon clipping

Learning objectives

Core assignment: to understand and be able to program a sequence of cubic splines; to be able to animate a number of objects moving smoothly along those parametric curves.
Completion assignment: to demonstrate an understanding of the use of rotation transformations; to demonstrate use of the calculation of closest distance between a line and a point.
Extension assignment: to show understanding of a basic polygon clipping algorithm; to demonstrate the ability to implement a user interface that allows the user to select and move one of several points.

Practical matters

You should submit up to three sketches: one for core, one for completion, and one for extension. Each sketch will be in its own directory. You will need to use the zip utility to put all the directories into a single zip file and upload that one zip file. You then upload that one zip file to the ECS submission system. For example, if you complete all three sketches, and give them the names you are told to give them, then you can run this command to make the zip file:

```
zip -r A3submission.zip A3Core A3Comp A3Extn
```

The "`-r`" option tells zip to include all of the files inside the directories. The file that you submit is `A3submission.zip`. See the course website for details of how to use the submission system.

Your sketches must run on the version of Processing used on the ECS machines without needing to be modified.

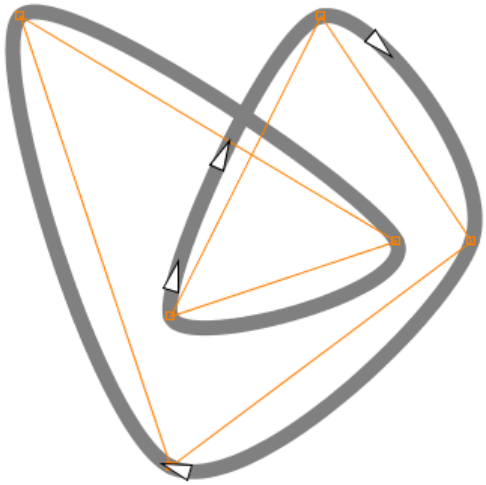
Marking

Core assignment: completing this satisfactorily can provide up to **60%** of the marks.
Completion assignment: completing this will provide a further **20%** of the marks.
Extension assignment: completing this will provide a further **10%** of marks.
Code quality: the final **10%** of the marks will be awarded based on the quality of your code.

Core assignment — looping along a path

You are required to write code that will draw a closed loop of Catmull-Rom cubic splines defined by six control points.

The image at right shows an example screen shot, with the Catmull-Rom spline in thick lines, the six control points identified by small boxes, and the boxes connected by thin lines to show the order of the control points.



Hint: Catmull-Rom splines are implemented in Processing using the `curve()` and `curveVertex()` functions.

As explained in lectures, for a sequence of Catmull-Rom cubic splines, each spline is defined by four control points and any one spline has three control points in common with the next spline in the loop.

Next, animate a number of objects moving along the Catmull-Rom spline. The objects must move smoothly along the curve, must have a shape whose orientation can be easily seen, and must be oriented to point in the direction of travel.

You can use your imagination in the design of the moving objects. The triangles shown in the example are just the simplest possible object with easily identified orientation.

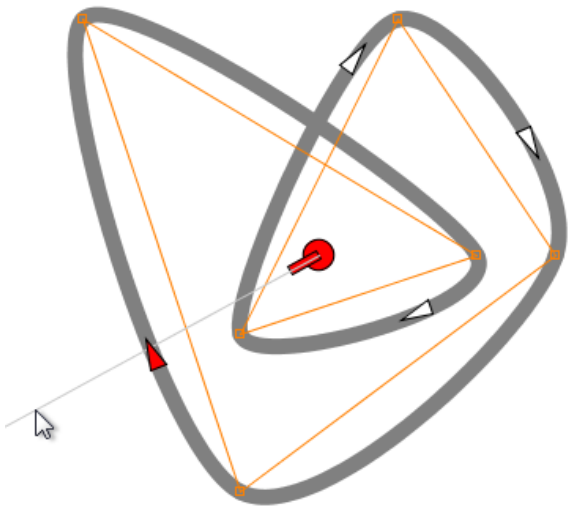
Hint: You will ideally use `translate()` and `rotate()` to put the objects in the correct location at the correct orientation. You should investigate the `curvePoint()` and `curveTangent()` functions, which are the Catmull-Rom equivalent to the Bézier `bezierPoint()` and `bezierTangent()` functions that you learnt about in Worksheet 3.

Save this sketch as `A3Core`.

Completion assignment — laser pointer

Save a fresh copy of your core sketch, as `A3Comp`, and modify it as follows.

Draw a laser pointer in the center of the screen that always looks as the mouse, with a laser line extending from the laser pointer towards the mouse (and beyond).



The laser pointer should not be on the spline path, and have a shape whose orientation can be easily seen.

Next, highlight any objects that are moving along the spline that are close enough to the laser line. This could be done, for example, by drawing them in a different colour. To calculate the distance between the objects and the laser line, you can use the technique presented on slide 100 of the lecture slides.

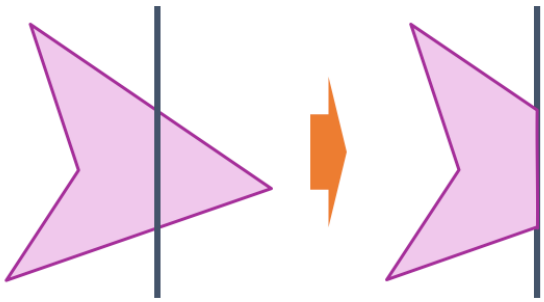
Save this sketch as `A3Comp`.

Extension assignment — clipping a polygon against an infinite line

The Sutherland-Hodgman polygon clipping algorithm clips an arbitrary polygon against an arbitrary convex clipping polygon. The algorithm was given, in outline, in the lectures.

The core of the algorithm clips an arbitrary polygon against a single infinite straight edge. Implement this part of the algorithm.

Implement a mechanism whereby the user is able to change the polygon by moving vertices. You will need to re-run the polygon clipping algorithm after every change to get the new clipped polygon. The clipping line does not need to be moveable.



Implement a way to demonstrate to your tutor that your algorithm does what it is supposed to do. A simple way to do that is to draw the original and the clipped polygon in different colours and to draw appropriate shapes at the vertices of both polygons, but feel free to think of other ways that you can visually demonstrate that your algorithm works.

Save this sketch as `A3Extn`.

Hints for completing the Extension assignment

Representing a polygon: The algorithm for clipping a polygon against an infinite straight line requires that the inputs be the infinite clipping line and the polygon to be clipped and that the output be the clipped polygon. A good way to represent a polygon is as an `ArrayList` of `PVector`. Ideally, you would wrap this data structure in a class.

Generating the output polygon: The algorithm takes a polygon as input and outputs the clipped polygon. Given that the assignment requires you to display both the input polygon and the output polygon at the same time, you may find it most helpful to generate a completely new polygon when doing the clipping.

Infinitely-long clipping line: The best way to represent the infinite clipping line is by specifying two points that lie on that line. For drawing the infinite line, you can simply draw the line segment between the two ends, leaving the rest of the line to the user's imagination.