CS & MCS Section B
Michaelmas Term 2018

GEOMETRIC MODELLING

Practical 3
Bézier Curves

1 Practicalities about the practical session

This practical task will illustrate the calculation of a planar Bézier curve.

2 Getting the files

In order to get the necessary files copy everything from the appropriate directory:

```
% cp /usr/local/practicals/gmod/Bezier/* .
```

To compile your code (after making appropriate changes), you can use the command

```
% javac MyPolygon.java
```

and to run it, use

`% appletviewer Bezier.html &` or `% java  Bezier &`

When running the code, you are presented with a window. Clicking the left mouse button over the window defines a new point. A few left clicks will generate the (planar) polyline on which the control points for the Bézier curve lie.

You may find it equally easy to input points by reading them from a file rather than clicking the mouse across the window. The button *Read sample points* assumes a file called `sample.data` exists in the current directory and causes a set of integer coordinate pairs to be read in order from this file. A new polyline is generated as a result; the Bézier degree elevation algorithm will run on this data set.

The button *Clear all* clears the screen and allows you to start a new polyline. The action expected when clicking the button *Elevate once* is that a *degree elevation* of the polyline you have just input should be computed and drawn. When the button *Elevate once* is pressed repeatedly, successive degree elevations should be computed and displayed.

3 What you need to do

The goal of this practical is to draw a Bézier curve defined by the control points input by the user. You are asked to use *degree elevation* in order to calculate the points on the curve.

The control polygon is stored in `MyPolygon`; its degree–elevated version will be stored in the related `Polygon` structure[1] `Elevated`. The number of initial control points is `npoints`, i.e. the number of points in `MyPolygon`. The number of points in `Elevated` is `Elevated.npoints` and gets incremented with each elevation.

Task 1 On the way to drawing the curve, you will first want to calculate the points for just one degree elevation. The function which corresponds to this operation is called `elevateOnce()`.

It will be useful to display both polylines on the screen, so as to check that the curve is really the one determined by the given control points. In order to do this, preserve the original control points, and modify only the curve points. you can do this 'in place' or using a temporary array for the calculations.

The new set of curve points can be calculated from the current set of curve points according to the formula

$$P_i^{\text{new}} = \frac{i}{n} P_{i-1}^{\text{old}} + \frac{n-i}{n} P_i^{\text{old}}$$

where n is the current number of points in the curve, i is the index of the point whose coordinates are being calculated and P_i are the actual points.

When implementing the formula above, you need to take care of two essential details:

- one is the way you choose the variation range and direction for i so as not to overwrite values you need at the next step;
- the other is the way in which you avoid causing 'integer division' when computing the fraction $\frac{i}{n}$ (e.g. by casting one of the operands to `double`).

For improved accuracy, you should do all the calculations in double precision and only round the results back to integer coordinates before you return the result. (This is already partially done for you.)

You will also need to check whether it is the first time you perform a degree elevation on a given set of control points. If so, the curve points will need to be initialised to the control points before the first degree elevation is performed. The `boolean` flag `elevated` is provided for this purpose.

Once your function, prompted by the user pressing the appropriate button, has calculated the new set of curve points, the corresponding polyline will be displayed automatically.

[1] Details of AWT Polygon at `http://download.oracle.com/javase/1.5.0/docs/api/java/awt/Polygon.html`

Task 2 Now draw the refined Bézier curve by repeated degree elevation. Your program should compute increasingly refined versions of the curve each time the user presses the button labelled *Elevate once*. It would be helpful if the program printed the degree number on each elevation.

Task 3 (*optional*) Once you have the degree elevation routine working, try calling it a preset number of times, by defining a number of degree elevations. Suggest a suitable value that gives a smooth curve in most cases.

Note that integer arithmetic will not be accurate enough for the calculations, as too much information is lost in the process. Use a couple of floating point arrays `double tempX[]` and `double tempY[]` to store the intermediary results at each elevation. These have been set up so that they can be extended with the method `resizeTemp()`. Only when you have completed the cycle of degree elevations it is advisable to round each coordinate of the temporary points to the nearest integer in order to store it in the curve points and display it.

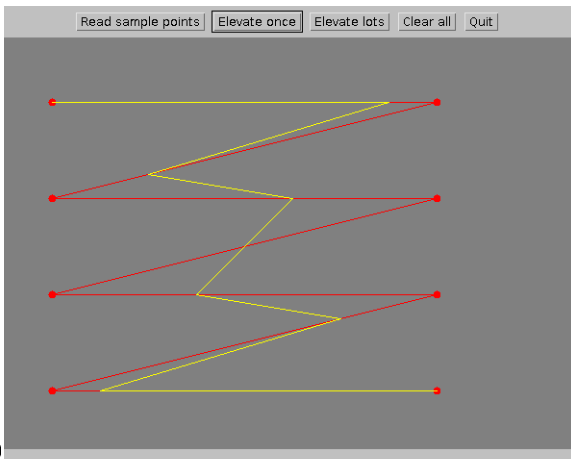
You will also need to modify the graphics routines in `Bezier.java` in order to define a new button to trigger this function.

Task 4 (*optional*) Find a way to stop the degree elevation process dynamically, without having to give a predefined value for the number of degree elevations.

Task 5 (*optional*) Implement mouse listeners that would enable the user to grab a control point, thus causing the curve to be recomputed and redrawn at its new location.

You will need screen-shots of your results.. Use `Applications -> Accessories -> Take Screen Shot` to help with this.

Irina Voiculescu
irina@cs.ox.ac.uk

(a)


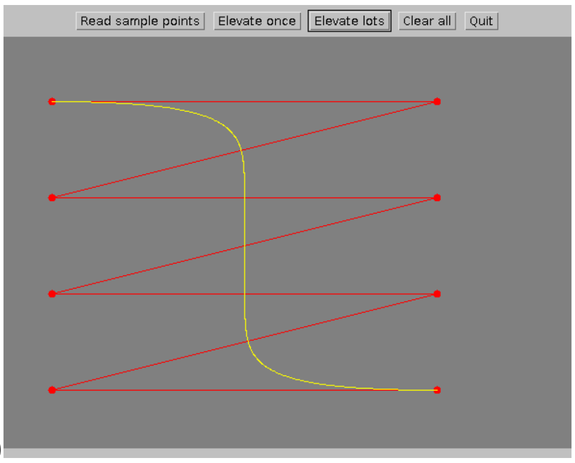
(b)


Figure 1: Polylines and the result of (a) one or (b) many degree elevations