Introduction to Programming

—

Midterm examination on machine

—

G1: clement.riu(at)enpc.fr G2: belos.liza(at)gmail.com G3: pascal.monasse(at)enpc.fr
G4: emanuele.concas(at)enpc.fr G5: thomas.daumain(at)enpc.fr G6: laura.echeverri-guzman(at)enpc.fr
G7: niloufar.fulami(at)aol.com

—

28/10/22

—

1 Instructions

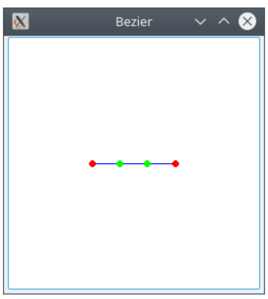
1.1 Bézier Curves

A cubic Bézier curve is a parametric curve used a lot in computer graphics. The curve $P(s)$, $(0 \leq s \leq 1)$, is parameterized by 4 control points $P_0 \dots P_3$:

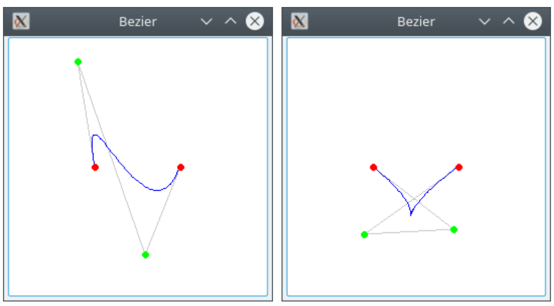
$$P(s) = (1-s)^3 P_0 + 3(1-s)^2 s P_1 + 3(1-s)s^2 P_2 + s^3 P_3. \tag{1}$$

Note that $P(0) = P_0$ and $P(1) = P_3$ but P_1 and P_2 are not on the curve in general. Our program explores the various shapes the curve can take. Without loss of generality, we will assume $P_0 = (0,0)$ and $P_3 = (1,0)$.

It is more important to deliver a clean code (commented and correctly indented) that *compiles* than answering all questions. For that, check after each step that the build works. Write in comments of the code the question number. At the end, create an archive with source code and file `CMakeLists.txt` to upload on educnet.



Initial Bezier curve



Two Bezier curves after motion of the control points. Notice the cusp in the right image.

```
bool track(int& x, int& y) {
  Event e;
  while(true) {
    getEvent(-1,e);
    if(e.type == EVT_BUT_OFF)
      return false;
    if(e.type == EVT_MOTION) {
      x = e.pix[0];
      y = e.pix[1];
      return true;
    }
  }
}
```

Function used in question 15.

1.2 Point structure

1. Create an Imagine++ project. In a file *separate* from the one containing your `main` function, define the structure `point` taking coordinates of type `float`.

2. Define the operators for addition and subtraction of two points.
3. Define the operators for multiplication and division by a float. For multiplication, define the two operators: float *`point` and `point`* float.
4. Define a function `affine` taking a point and applying the similarity $P \to a * P + S$ with parameters a the zoom factor and S a point (shift). It will be used to map standard coordinates of the curve to pixel coordinates for drawing.
5. Define a function `rotate` that rotates a point P around a center C with an angle α expressed in degrees. Functions cos and sin from #include <cmath> take their argument in radians.

1.3 Bézier curve

6. In a new separate file, define a structure `Bezier` storing an array of four points.
7. Write a function `initBezier` that returns a new curve with $P_1 = (1/3, 0)$ and $P_2 = (2/3, 0)$.
8. To draw the curve, we will zoom and shift the coordinates for display. The window will be square with `dim`×`dim` pixels, point P_0 will be displayed at (`dim`/3,`dim`/2) and P_3 at (2`dim`/3,`dim`/2). Values of s will be discretized uniformly at `npoints`= 100 values. Define the adequate constants, `dim`= 512.
9. The function `draw` takes a curve and displays it: draw a blue line from $P(s)$ to $P(s + \delta s)$ (use `affine` to apply the transform) with $\delta s = 1/$`npoints`.
10. Add in the previous function the display of the 4 points (red for extremities, green for P_1 and P_2), disks of `radius`= 3 pixels.
11. Make the `main` function display an initial Bezier curve and wait for a click to continue.

1.4 Animation

12. Write a function `animate` in the main file that applies 100 iterations of rotation around P_0 of $P_1 = (3/4, 0)$ and of $P_2 = (2/3, 0)$ around P_3. At iteration i, P_1 rotates of $i * 10°$ and P_2 of $i * 3°$. After each display, a small pause is observed. At the end, the function should wait a point click.
13. Enrich the function `draw` by linking the control points by lines in gray color of intensity 200. This should be done at the beginning of the function, so that it does not overlap with the rest.

1.5 User interaction

14. Write a function `selectPoint`, taking a curve and waiting until the user clicks on P_1 or P_2. A right click exits and returns false, whereas a left click stores the pixel coordinates of the click and the point number selected. While the left click occurs elsewhere, it continues waiting for a click. Be careful that the click coordinates are in pixels while the points are around the interval $[0, 1]$, so that an affine transform must be applied to compare.
15. Write a function `interactive`. It displays an initial Bézier curve and loops undefinitely until `selectPoint` returns with a right click. Inside the loop, it lets the user move the selected controlled point and displays interactively the curve. The function `track` (see figure) is used to detect a mouse motion or the mouse button release. In case of motion, the shift from the previous position in pixels is applied to the control point. Be careful that the scale of display is not the same as the point coordinates.
16. *Bonus*: (i) Move the interactive control in a separate program (within the same project CMakeLists.txt), and (ii) create a library for the common functions of the two programs.