

Lab 1 - 2D Bézier Curves

Laura Mazzuca - matr. 0000919489

04/09/2021

0 Assignments

1. Draw Bézier curve with the de Casteljau algorithm
2. Drag control points to change their position
3. Draw Bézier curve with the optimized adaptive subdivision method

1 De Casteljau algorithm

The de Casteljau algorithm was implemented in the *deCasteljau()* function taking in input t as parameter value, an *inArray* containing the control points coordinates, an *outArray* containing the algorithm's results (i.e. the curve points) and the *size* of the inArray array.

To support the algorithm, the *lerp()* function was implemented in its 1D version.

The number of curve points drawn is 100 and the resulting curve can be seen in Figure 1.

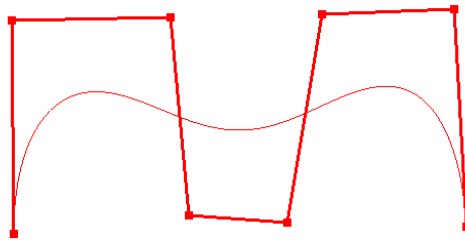


Figure 1: Curve generated with the de Casteljau algorithm.

2 Drag control points to change their position

This functionality was implemented by checking if the clicked point has already a control point in the *isControlPoint()* function. If yes, this function returns the index of the control point hit, which gets saved in the *SelectedCP* global variable. Then, the function *myMotionFunc()* handles the logical and graphical update of the point.

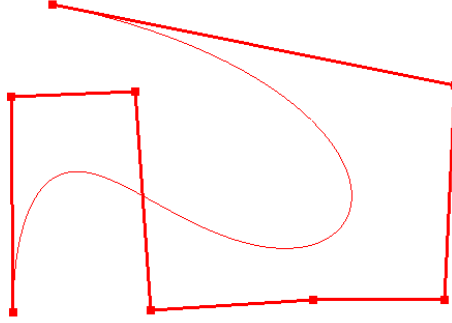


Figure 2: Same points as in Figure 1, but re-positioned.

3 Optimized adaptive subdivision method

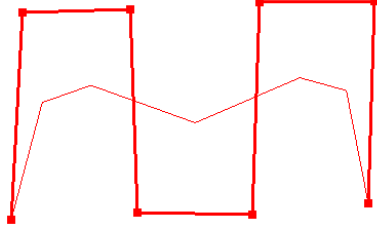
This algorithm was implemented in the recursive function *adaptiveSubdivision()*, which takes in input a *tempArray* of control points and its *size*.

The **Adaptive Subdivision** algorithm is a recursive algorithm that selects the curve points to draw from a set of control points obtained from splitting the original Bézier curve in two sub-curves of the same degree. To obtain the control points for both the sub-curves, the function *deCasteljauASCP()* was implemented, which saves in two pre-prepared arrays the intermediate results of the de Casteljau algorithm. Namely, the first one, which should contain the first half of the curve, is populated with the p_0^n results while the second one is populated with the p_1^n results.

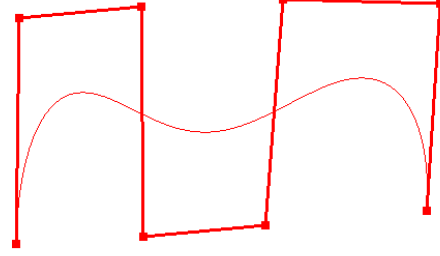
Then, for each of the resulting sub-curves we need to check if it passes the so-called Flat Test. The **Flat Test**, checks that the distance between each of the intermediate control points in the current group and the segment resulting from the first and last control points is less than a certain *Tolerance* value. If the check is passed then the current sub-curve can be approximated with the aforementioned segment and the control points can be added to the curve points, otherwise it can't and the algorithm needs to subdivide the sub-curve once again.

To support all the steps for the Flat Test, the following algorithms were implemented using the *glm::vec3* to ease input and output and to use some built in glm function:

- *sub()*, which computed the subtraction between two vectors;
- *magnitude()*, which computed the magnitude of a vector;
- *distPointLine()*, which computed the distance between a point and a segment.



(a) *Tolerance* = 0.075.



(b) *Tolerance* = 0.00075.

Figure 3: The visual results of the Adaptive subdivision algorithm with different Tolerance values.