Bezier Curves

Octave Project 3

# Description of the problem:

The mathematical problem we are trying to solve with these methods is interpolating 2D or 3D curves using a set of control points. Bezier curves are defined as a linear combination of Bernstein polynomials as follows. At the end we are using those control points to create the Bernstein polynomials that define the curve.

# Explanation of the method:

**Direct Evaluation:**

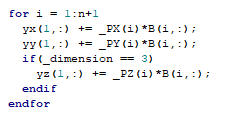
Given points we will compute the Bernstein polynomials where is , we will have a mesh of t values in . Finally compute the Bezier curve . Normally we will compute the curve first and then compute for each of the t values. In the code we first compute the values of the Bernstein polynomials by plugging the t values for each one of them.

A picture containing text

Description automatically generated

Figure

In *Figure 1* we can se the computation of each of the Bernstein polynomials, in the code the t represents an array of values from .



Figure

In *Figure 2* we are computing .

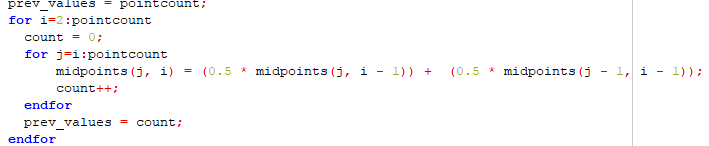
**De Casteljau:**

Given P0, P1 … Pn points in or the Bezier curve can be computed constructing a regular mesh of m + 1 nodes in [0, 1] and then applying linear interpolation recursively on for each node. The recursion is noted as follows:

Texto, Carta

Descripción generada automáticamenteAnd in code it looks like this

**Midpoint:**

****Given P0, P1 … Pn points in or the Bezier curve can be split into two different curves, starting at P0  and ending at where , hence the name of midpoint subdivision, and starting at  and ending at Pn. Using De Casteljau evaluating at  we can compute the first midpoint:

From this we can use , , , as the new control points to construct the curve , which can be defined by the following formula:



Similarly, can also be constructed by using , , , as control points, which can be defined by the following formula:

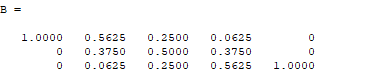


Here leftDiv and rightDiv will contain the control points for and . Furthermore, we can keep subdividing the curves and to approximate our curves with more precision and smoothness. Code implementation wise this algorithm has exponential growth which is something to avoid, that is why when implemented usually is restricted to a certain amount of iterations for the subdivision, in this case the variable maxIT represents that while currIt keeps track of the current iteration in which we are.

# Examples:

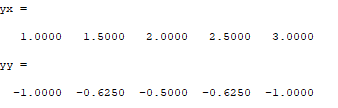
**Direct Evaluation:**

Given and a mesh of 5 nodes in [0,1] , we compute .

We check the results with the ones from the implemented code 

We can see that the output is the same. We can now compute the points for the x and y coordinate with

We check the results with the ones from the implemented code



We can see that the results are the same (each column represents one ).

**De Casteljau:**

Given .

For :

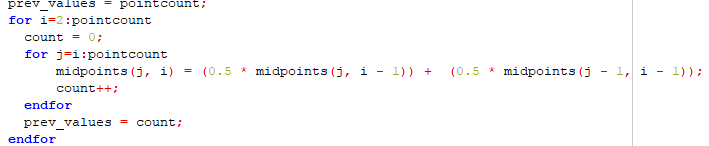
For :

For :

Gráfico, Gráfico de líneas

Descripción generada automáticamenteGráfico, Gráfico de líneas

Descripción generada automáticamenteOutput of the program compared with the one obtained by hand.

**Midpoint Subdivision:**

Given: and performing two iterations of the algorithm we get the following results:

1st Iteration:

Points in after the iteration: {(0, -1), (), (1, 3) }.

2 new curves can be created: and with control points:

: {(0, -1), (1, ), ()}

: {(), (), (1, 3)}

2nd Iteration:

For given the control points :

For given the control points :

Points in after the iteration: {(0, -1), (), (), (), (1, 3)}.

4 new curves can be created with control points:

: {(0, -1), (), ()}

: {), (), ()}

: {(), (), ()}

: {(), (), (1, 3)}

# Bibliography:

MAT300 Lecture Notes: lecture11, lecture 12.