Project 3

Kerman Munitxa, Júlia Gumà, Markel Pisano
kerman.m@digipen.edu j.gum@digipen.edu markel.p@digipen.edu
540002517 540000117 540002615

1. Direct evaluation

1.1 Description of the problem

Given some control points, it creates a Bezier curve. This curve passes through the first and the second point, while the other points control its curvature. The curve is created by using direct evaluation of Bernstein polynomials.

1.2 Mathematical explanation of numerical methods

Given $n+1$ control points in $\mathbb{R}^2$ or $\mathbb{R}^3$, a Bezier curve is a polynomial $\gamma : [0,1] \longrightarrow \mathbb{R}^2$ or $\mathbb{R}^3$ defined as:

$$\gamma(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{1}$$

Where B_i^n is the i^{th} Bernstein polynomial of degree n defined as:

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \qquad \text{for } i = 0, 1, \ldots, n \tag{2}$$

We create a regular mesh of nodes in $[0,1]$, where $t_j = \frac{j}{m}$ for $j = 0, 1, 2, \ldots, m$. Doing this we are making sure that there is the same distance between nodes, and that there are no repeated values of t.

For each node t_j we evaluate the Bernstein polynomials $B_i^n(t_j)$ for $i = 0, 1, \ldots, n$ and store the values. For each node we compute the sum $\gamma(t) = \sum_{i=0}^{n} P_i B_i^n(t)$ and store the values.

With this, we already have the values for the corresponding axis in the specific t_j, so we just have to plot it.

1.3 Code implementation

In the code, first of all we create the mesh of nodes (line 25). The vectors *result_x*, *result_y* and *result_z* (line 28) are used to store the resulting points of the curve. We iterate over all the control points(line 38). Here we obtain the correspondant Bezier polynomial of this point, so we evaluate it at the corresponding value of t(line 46). Now we multiply this array by the current control point and we add it to the final result (line 49). Lastly, we just plot the points and the curve (line 60).

1.4 Examples

Let's take for example the points $P_0 = (0,3), P_1 = (0,-2), P_2 = (2,1)$ and a mesh of four nodes, which would be $t_0 = 0, t_1 = \frac{1}{3}, t_2 = \frac{2}{3}$ and $t_3 = 1$. Having 3 points means out $n = 2$. Out Bernstein polynomials will be:

$$B_0^2(t) = \binom{2}{0}(1-t)^{2-0}t^0 = t^2 - 2t + 1$$

$$B_1^2(t) = \binom{2}{1}(1-t)^{2-1}t^1 = -2t^2 + 2t$$

$$B_2^2(t) = \binom{2}{2}(1-t)^{2-2}t^2 = t^2$$

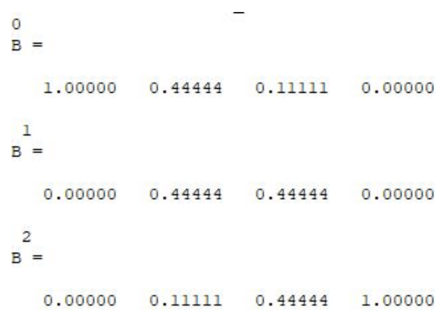
Evaluated at our mesh of t, Bernstein values will be:

$B_0^2(t) = \{1, \frac{4}{9}, \frac{1}{9}, 0\}$

$B_1^2(t) = \{0, \frac{4}{9}, \frac{4}{9}, 0\}$

$B_2^2(t) = \{0, \frac{1}{9}, \frac{4}{9}, 1\}$

Compared to the obtained values:



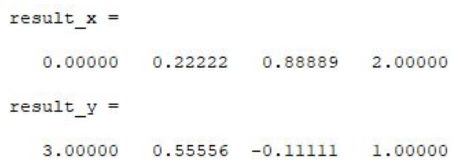
```
 0
B =

   1.00000   0.44444   0.11111   0.00000

 1
B =

   0.00000   0.44444   0.44444   0.00000

 2
B =

   0.00000   0.11111   0.44444   1.00000
```

As we can see, they are equals. Now we multiply all of the values on each axis and sum them:

x-axis: $0\{1, \frac{4}{9}, \frac{1}{9}, 0\} + 0\{0, \frac{4}{9}, \frac{4}{9}, 0\} + 2\{0, \frac{1}{9}, \frac{4}{9}, 1\} = \{0, \frac{2}{9}, \frac{8}{9}, 2\}$

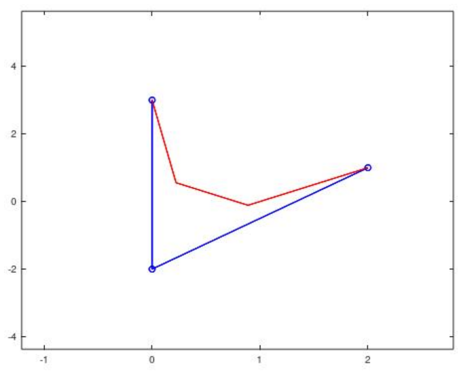
y-axis: $3\{1, \frac{4}{9}, \frac{1}{9}, 0\} - 2\{0, \frac{4}{9}, \frac{4}{9}, 0\} + 2\{0, \frac{1}{9}, \frac{4}{9}, 1\} = \{3, \frac{5}{9}, -\frac{1}{9}, 1\}$

Compared with the octave results:



```
result_x =

   0.00000    0.22222    0.88889    2.00000

result_y =

   3.00000    0.55556   -0.11111    1.00000
```

Now if we plot the curve using the resultant points:



1.5 Observations

Incrementing the number of outputnodes (the length of the mesh) does not increment much the computing time. This is probably due to using the mesh with array operators and octave is powerful with it. However, if we increment the cumber of control points (the n of the problem), the time increases in a considerable way, probably because we increment the iterations inside the loop, where there are a bunch of multiplications and the function for finding the combinatory number, what might be expensive. This time has a linear form. Also, if we increment a lot the number of control points, octave throws a warning because the function for getting the combinatory number loses precision. For all time calculations, plotting was not taken into account, since it is not part of the mathematical field.

2. De Casteljau

2.1 Description of the problem

Given some data, the *De Casteljau* algorithm applies recursively linear interpolation to obtain $\gamma(t_j)$ at the values of the mesh.

2.2 Mathematical explanation of numerical methods

Given $n+1$ control points in $\mathbb{R}^2$ or $\mathbb{R}^3$, we create a mesh of nodes in $[0,1]$, where $t_j = \frac{j}{m}$ for $j = 0, 1, 2, \ldots, m$. Doing this we are making sure that there is the same distance between nodes, and that there are no repeated values of t.

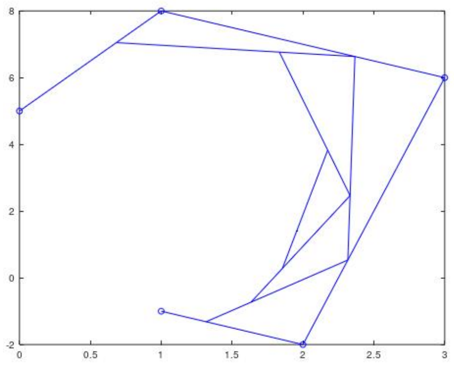
For each node we have to apply recursively linear interpolation in order to obtain $\gamma(t_j)$. This is done through the following formula:

$$P_i^k(t_j) = t_j P_{i+1}^{k-1} + (1 - t_j) P_i^{k-1} \tag{3}$$

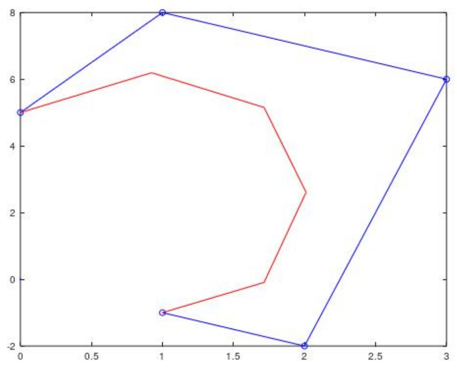
For every t_j, we obtain the $\gamma(t_j)$ in the following way:

$$\begin{array}{ccccc}
\text{Level i=0} & i=1 & i=2 & \cdots & i=n \\
P_0 & & & & \\
 & P_0^1 & & & \\
P_1 & & P_0^2 & & \\
 & & & \ddots & \\
\vdots & & & & P_0^n = \gamma(t_j) \\
 & & P_{n-2}^2 & & \\
\vdots & P_{n-1}^1 & & & \\
P_n & & & &
\end{array} \tag{4}$$

If we do a polygonal line for each level, we would see a graph like the following one:



With all the $\gamma(t_j)$, we obtain the points that create the resulting curve, shown in red in the following graph:



The more nodes we use, the more precise and smooth the curve will be.

2.3 Code implementation

In the code, first of all we create the mesh of nodes (line 25). The vectors *result_x*, *result_y* and *result_z* (line 40) are used to store the resulting points of the curve, obtained after iterating through each level. For every t_j, we start with the control points and, in every iteration for that t_j, we compute the new vector of values (line 64), which will have the length of the previous vector of values minus one. The new vector of values will be the new base for computing the following vector of values. We do this until the vector has only one element, which is the $\gamma(t_j)$. When we finish one level, we restart the vector of values to the control points (line 113) and restart the process with the new t_j. Once we have all the $\gamma(t_j)$, we display the given points (line 127 and 148) and the curve (line 134 and 155). After that, we display the shells at a specific given index (line 138 and 159).

2.4 Examples

Let's take for example the points $P_0 = (0, 3), P_1 = (1, -2), P_2 = (4, 2)$ and a mesh of four nodes, which would be $t_0 = 0, t_1 = \frac{1}{3}, t_2 = \frac{2}{3}$ and $t_3 = 1$.

For $t_0 = 0$ we would obtain:

$$\begin{array}{ccc} \mathrm{P}_0 = (0,3) & & \\ & P_0^1 = (0,3) & \\ P_1 = (1,-2) & & \gamma(0) = (0,3) \\ & P_1^1 = (1,-2) & \\ P_2 = (4,2) & & \end{array} \tag{5}$$

For $t_1 = \frac{1}{3}$ we would obtain:

$$\begin{array}{ccc} \mathrm{P}_0 = (0,3) & & \\ & P_0^1 = (\frac{1}{3}, \frac{4}{3}) & \\ P_1 = (1,-2) & & \gamma(\frac{1}{3}) = (\frac{8}{9}, \frac{2}{3}) \\ & P_1^1 = (2, -\frac{2}{3}) & \\ P_2 = (4,2) & & \end{array} \tag{6}$$

For $t_2 = \frac{2}{3}$ we would obtain:

$$\begin{array}{ccccc} \mathrm{P}_0 = (0,3) & & & & \\ & & P_0^1 = (\frac{2}{3}, -\frac{1}{3}) & & \\ P_1 = (1,-2) & & & & \gamma(\frac{2}{3}) = (\frac{20}{9}, \frac{2}{3}) \\ & & P_1^1 = (3, \frac{2}{3}) & & \\ P_2 = (4,2) & & & & \end{array} \tag{7}$$

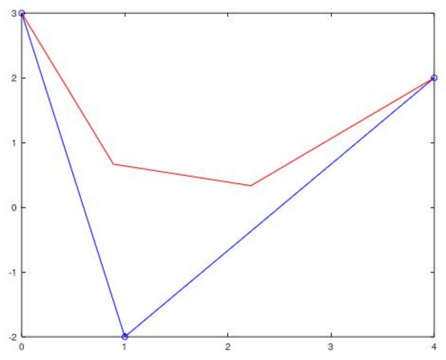
For $t_3 = 1$ we would obtain:

$$\begin{array}{ccccc} \mathrm{P}_0 = (0,3) & & & & \\ & & P_0^1 = (1,-2) & & \\ P_1 = (1,-2) & & & & \gamma(1) = (4,2) \\ & & P_1^1 = (4,2) & & \\ P_2 = (4,2) & & & & \end{array} \tag{8}$$

In Octave we get these results:

- For $t_0 = 0$: $vec\_x = 0 \quad vec\_y = 3$
- For $t_1 = \frac{1}{3}$: $vec\_x = 0.88889 \quad vec\_y = 0.66667$
- For $t_2 = \frac{2}{3}$: $vec\_x = 2.2222 \quad vec\_y = 0.3333$
- For $t_3 = 1$: $vec\_x = 4 \quad vec\_y = 2$

With these points, we obtain the following curve:



2.5 Observations

If we input a huge amount of output nodes, the resulting graph will lose precision due to floating point errors, apart from taking a lot of time. One advantage of this method is that inserting a new point is really easy and doesn't mean to recompute everything.

3. Midpoint subdivision

3.1 Description of the problem

Given $n+1$ points, $\gamma(\frac{1}{2})$ is computed through *De Casteljau* algorithm to obtain two new curves and keep iterating finding $\gamma(\frac{1}{2})$ for the new control points.

3.2 Mathematical explanation of numerical methods

Similar to De Casteljau's algorithm, points along curve γ are computed through linear interpolation of control points $S = \{P_0, \ldots, P_n\}$. The difference is the only t value evaluated along the curve is $\gamma(\frac{1}{2})$. To evaluate $\gamma(\frac{1}{2})$:

$$\begin{array}{ccccc}
\text{Level i=0} & i=1 & i=2 & \cdots & i=n \\
P_0 & & & & \\
 & P_0^1 & & & \\
P_1 & & P_0^2 & & \\
 & & & \ddots & \\
\vdots & & & & P_0^n = \gamma(\frac{1}{2}) \\
 & & P_{n-2}^2 & & \\
\vdots & P_{n-1}^1 & & & \\
P_n & & & &
\end{array} \tag{9}$$

Each intermediate points is computed by linearly interpolating element's of the previous level i as such:

$$P_j^i(t) = tP_{j+1}^{i-1} + (1-t)P_j^{i-1} \quad \text{with } i = 1, \ldots, n \text{ and } j = 0, \ldots, n-i \tag{10}$$

Once the pyramid above is computed for every intermediate point, control points for the next iteration are selected for curves γ_1 and γ_2:

Curve γ_1 lies in hull defined by $P_0, P_0^1, \ldots, P_0^{n-1}, P_0^n$

Curve γ_2 lies in hull defined by $P_0^n, P_1^{n-1}, \ldots, P_{n-1}^1, P_n$

So for defining γ_1 the "above" points P_0^i where $i = 0, \ldots, n$ are taken. And for γ_2 the "below" points P_i^{n-i} where $i = 0, \ldots, n$ are taken.

The process is repeated for k iterations, computing $\gamma(\frac{1}{2})$ at each iteration with the new control points.

2^k curves are computed.

$2^k + 1$ points define the final curve.

3.3 Code implementation

Through the recursive function *midpoint_recursive* every point in the curve is calculated (line 20). This takes as parameters *it*(the current iteration), *max_it*(number of iterations to be done) and control points (*PX,PY,PZ*). It outputs a single point ($\gamma(\frac{1}{2})$) in the case of the last iteration, or a combination of every resultant point (line 135) in the splited curves (γ_1 and γ_2) and the midpoint of the current curve ($\gamma(\frac{1}{2})$) as $[\gamma_1, \gamma(\frac{1}{2}), \gamma_2]$.

For creating the pyramid and the intermediate points belonging to it and to the splited curves as their hull (control points), a double nested loop is used (line 86). First outter loop iterates through level $i = 1$ to $i = n$ and inner loop iterated through points of current level $j = 0$ to $j = n - i$, being $n + 1$ the number of control points.

Intermediate points of the current iterated level are stored in a local variable *prev* (line 102), for then add the first element to the control points of curve γ_1 (line 107) and the last to γ_2 (line 111).

If the algorithm is in the last iteration, it plots the control points (line 149).

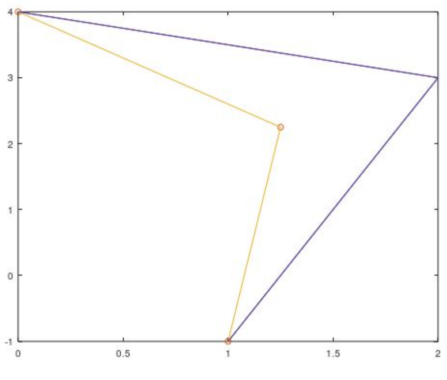
Sanity checks are made (line 125) to ensure that outputed point count is equal to $2^k + 1$, where k is *miditerations.*

3.4 Examples

Let $P_0 = (1, -1), P_1 = (2, 3)$ and $P_2 = (0, 4)$ be our control points.

- 1st iteration, curve γ_0
  For γ_0

$$\begin{array}{ccc} P_0 = (1,-1) & & \\ & P_0^1 = (\frac{3}{2}, 1) & \\ P_1 = (2,3) & & \gamma_0(\frac{1}{2}) = (\frac{5}{4}, \frac{9}{4}) \\ & P_1^1 = (1, \frac{7}{2}) & \\ P_2 = (0,4) & & \end{array} \tag{11}$$



(i) 1st iteration

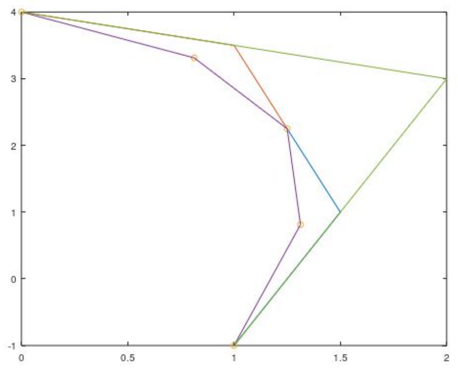
Octave's output points:
x = 1.00000 1.25000 0.00000
y = -1.0000 2.2500 4.0000

- 2nd iteration, curves γ_1 and γ_2:
  For γ_1

$$\begin{array}{ccc} P_0 = (1,-1) & & \\ & P_0^1 = (\frac{5}{4}, 0) & \\ P_1 = (\frac{3}{2}, 1) & & \gamma_1(\frac{1}{2}) = (\frac{21}{16}, \frac{13}{16}) \\ & P_1^1 = (\frac{11}{8}, \frac{13}{8}) & \\ P_2 = (\frac{5}{4}, \frac{9}{4}) & & \end{array} \tag{12}$$

For γ_2

$$\begin{array}{ccc} P_0 = (\frac{5}{4}, \frac{9}{4}) & & \\ & P_0^1 = (\frac{9}{8}, \frac{23}{8}) & \\ P_1 = (1, \frac{7}{2}) & & \gamma_2(\frac{1}{2}) = (\frac{13}{16}, \frac{53}{16}) \\ & P_1^1 = (\frac{1}{2}, \frac{15}{4}) & \\ P_2 = (0, 4) & & \end{array} \tag{13}$$



(i) 2nd iteration

Octave's output points:
x = 1.00000 1.31250 1.25000 0.81250 0.00000
y = -1.00000 0.81250 2.25000 3.31250 4.00000

- 3rd iteration, curves $\gamma_3, \gamma_4, \gamma_5$ and γ_6
  For γ_3

$$\begin{array}{ccc} P_0 = (1,-1) & & \\ & P_0^1 = (\frac{9}{8}, \frac{-1}{2}) & \\ P_1 = (\frac{5}{4}, 0) & & \gamma_2(\frac{1}{2}) = (\frac{77}{64}, \frac{-3}{64}) \\ & P_1^1 = (\frac{41}{32}, \frac{13}{32}) & \\ P_2 = (\frac{21}{16}, \frac{13}{16}) & & \end{array} \tag{14}$$

For γ_4

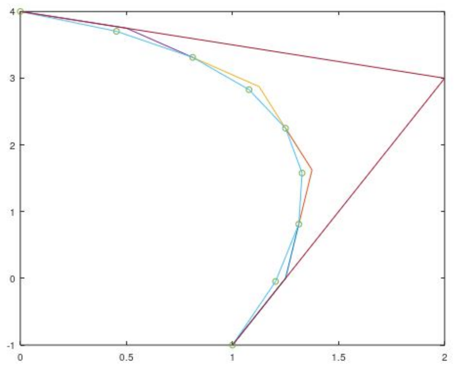
$$\begin{array}{ccc} P_0 = (\frac{21}{16}, \frac{13}{16}) & & \\ & P_0^1 = (\frac{43}{32}, \frac{39}{32}) & \\ P_1 = (\frac{11}{8}, \frac{13}{8}) & & \gamma_2(\frac{1}{2}) = (\frac{85}{64}, \frac{101}{64}) \\ & P_1^1 = (\frac{21}{16}, \frac{31}{16}) & \\ P_2 = (\frac{5}{4}, \frac{9}{4}) & & \end{array} \tag{15}$$

For γ_5

$$\begin{array}{ccc} P_0 = (\frac{5}{4}, \frac{9}{4}) & & \\ & P_0^1 = (\frac{19}{16}, \frac{41}{16}) & \\ P_1 = (\frac{9}{8}, \frac{23}{8}) & & \gamma_2(\frac{1}{2}) = (\frac{69}{64}, \frac{181}{64}) \\ & P_1^1 = (\frac{31}{32}, \frac{99}{32}) & \\ P_2 = (\frac{13}{16}, \frac{53}{16}) & & \end{array} \tag{16}$$

For γ_6

$$\begin{array}{ccc} P_0 = (\frac{13}{16}, \frac{53}{16}) & & \\ & P_0^1 = (\frac{21}{32}, \frac{113}{32}) & \\ P_1 = (\frac{1}{2}, \frac{15}{4}) & & \gamma_2(\frac{1}{2}) = (\frac{29}{64}, \frac{237}{64}) \\ & P_1^1 = (\frac{1}{4}, \frac{31}{8}) & \\ P_2 = (0, 4) & & \end{array} \tag{17}$$



(i) 3rd iteration

Octave's output points:
x = 1.00000 1.20312 1.31250 1.32812 1.25000 1.07812 0.81250 0.45312 0.00000
y = -1.000000 -0.046875 0.812500 1.578125 2.250000 2.828125 3.312500 3.703125 4.000000

3.5 Observations

Although it is a simple algorithm to implement, the biggest drawback of this method is the lack of precision to construct the mesh. Since it is a recursive algorithm, the mesh can only be of $2^k + 1$ nodes, where k is the iteration count selected.