Drawing Bezier Curves

1 General polynomial curves

Let $B(t)$ ($t \in [0,1]$) be a polynomial curve. A simple way to draw it would be to compute several points along it, like $B(0)$, $B(\Delta t)$, $B(2\Delta t)\dots B(N\Delta t) = B(1)$ (where $\Delta t = 1/N$) and join each pair of consecutive points with intervals. Thus, we need to evaluate the polynomial at a certain number of 'sample parameters' and join the resulting points with lines.

1.1 How to evaluate the polynomial?

Let's concentrate on scalar cubic polynomials. They are of the form

$$b(t) = at^3 + bt^2 + ct + d.$$

Now, the straighforward way to calculate $b(t)$ is by

$$b(t) = a * t * t * t + b * t * t + c * t + d.$$

It requires 6 multiplications and 3 additions (count the stars and pluses!). A better way is by

$$b(t) = ((a * t + b) * t + c) * t + d.$$

It takes only 3 additions and 3 multiplications.

1.2 Can we do it using even fewer multiplications?

We can if we know that what we need are the values of b at equispaced points, like $0, \Delta t, 2\Delta t, \dots, N\Delta t$ we mentioned using for polynomial curve drawing. The idea is to use finite differences

$$\begin{aligned} \Delta b(t) &= b(t + \Delta t) - b(t) \\ \Delta\Delta b(t) &= \Delta b(t + \Delta t) - \Delta b(t) \\ \Delta\Delta\Delta b(t) &= \Delta\Delta b(t + \Delta t) - \Delta\Delta b(t). \end{aligned}$$

It's not hard to see that these polynomials have degrees 2, 1 and 0. Thus, the last one is really a constant, independent of t.

Here's how to use the Δ's:

```
b := $b(0)$;

db := $\Delta b(0)$;

ddb := $\Delta\Delta b(0)$;

dddb := $\Delta\Delta\Delta b(0)$;

for i:=1 to N do

    b += db; // this is $b(i\Delta t)$!

    db += ddb;

    ddb += dddb;
```

Notice that this uses only 3 additions inside the loop; multiplications are needed only to initialize the variables (the first four lines). Notice that their number does not depend on N, the number of points we are computing.

1.3 Example

For $b(t) = t^3$:

$$\begin{aligned}\Delta b(t) = (t+\Delta t)^3 - t^3 = 3(\Delta t)t^2 + 3(\Delta t)^2 t + (\Delta t)^3 \\ \Delta\Delta b(t) = \Delta b(t+\Delta t) - \Delta b(t) = 6*(\Delta t)^2 t + 6(\Delta t)^3 \\ \Delta\Delta\Delta b(t) = 6*(\Delta t)^3.\end{aligned}$$

Thus, $\Delta\Delta\Delta b$ is indeed a constant and the other two are quadratic and linear.

2 Subdivision for Bezier curves

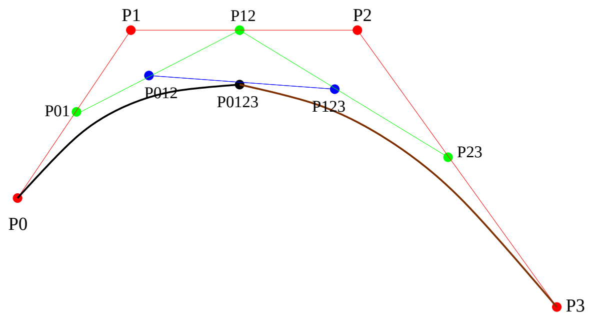

Figure 1: The Bezier curve with control points P_0, P_1, P_2, P_3 is the union of two Bezier curves with control points $P_0, P_{01}, P_{012}, P_{0123}$ and $P_{0123}, P_{123}, P_{23}, P_3$.

It turns out that the points we compute in the De Casteljeau algorithm when evaluating the Bezier curve at $t = .5$ can be used for the task of drawing the curve. More precisely, by computing (Figure 1)

$$\begin{aligned} P_{01} &= (P_0 + P_1)/2 \\ P_{12} &= (P_1 + P_2)/2 \\ P_{23} &= (P_2 + P_3)/2 \\ P_{012} &= (P_{01} + P_{12})/2 \\ P_{123} &= (P_{12} + P_{23})/2 \\ P_{0123} &= (P_{012} + P_{123})/2 \end{aligned}$$

we can split the Bezier curve to be drawn into two shorter curves. This procedure can be iterated to obtain a splitting into four, eight, sixteen, or even more short curves. These short curves are very well approximated by a polygonal curve joining the control points. Here is the pseudocode:

```
procedure draw_Bezier ( P0, P1, P2, P3 ):

    if enough subdivisions have been done then

        draw lines P0--P1, P1--P2, P2--P3 and return

    compute P01, P12, P23, P012, P123, P0123;

    draw_Bezier(P0,P01,P012,P0123);

    draw_Bezier(P0123,P123,P23,P3);
```

Notice that this procedure is extremely fast. It may seem to require divisions, but all those divisions are by two. Therefore they can be implemented as bit shifts (if integers are used) or as decrementing the exponent.