Quadratic splines

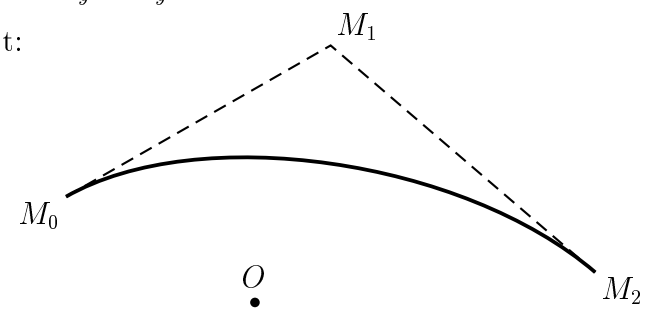
Alain Brobecker, februar 2010

The formulas for cubic splines ($at^3 + bt^3 + ct + d$) are well known, but it's also possible to go for quadratic splines ($at^2 + bt + c$). The latter will allow slightly faster incremental computation, but also easier computation of the roots of the formula $y = a_y t^2 + b_y t + c_y$, which can be handy for a scanline approach. The drawback is that we have only 3 control points, and we will loose fine control over the "power" of the derivatives.

So we are looking for a point $M(x(t); y(t))$ with quadratic coordinates:

$$\begin{cases} x(t) = a_x t^2 + b_x t + c_x \\ y(t) = a_y t^2 + b_y t + c_y \end{cases}$$

And we want that this point M is such that:



$$\begin{cases} \overrightarrow{OM}(0) = \overrightarrow{OM_0} \\ \overrightarrow{OM}(1) = \overrightarrow{OM_2} \\ \overrightarrow{OM}'(0) = \lambda \overrightarrow{M_0 M_1} & \lambda \geq 0 \\ \overrightarrow{OM}'(1) = \mu \overrightarrow{M_1 M_2} & \mu \geq 0 \end{cases}$$

$$\iff \begin{cases} c_x = x_0 \\ c_y = y_0 \\ a_x + b_x + c_x = x_2 \\ a_y + b_y + c_y = y_2 \\ b_x = \lambda(x_1 - x_0) \\ b_y = \lambda(y_1 - y_0) \\ 2a_x + b_x = \mu(x_2 - x_1) \\ 2a_y + b_y = \mu(y_2 - y_1) \end{cases} \overset{\text{see page 3}}{\iff \cdots \iff} \boxed{\begin{cases} c_x = x_0 \\ c_y = y_0 \\ b_x = 2(x_1 - x_0) \\ b_y = 2(y_1 - y_0) \\ a_x = x_2 - 2x_1 \\ a_y = y_2 - 2y_1 \end{cases}}$$

Like for the cubic splines, the coefficients are very simple, but we have only six of them instead of eight! Now if we want to compute n points of the spline incrementally we will use the method already described in the file *abrobecker.free.fr/text/squares.txt*. So we have:

$$\begin{aligned} x(0) &= x_0 \\ x(t+k) &= a_x(t+k)^2 + b_x(t+k) + c_x \\ &= a_x(t^2 + 2tk + k^2) + b_x t + b_x k + c_x \\ &= \underbrace{a_x t^2 + b_x t + c_x}_{x(t)} + \underbrace{2a_x tk + a_x k^2 + b_x k}_{inc1_x(t)} \end{aligned}$$

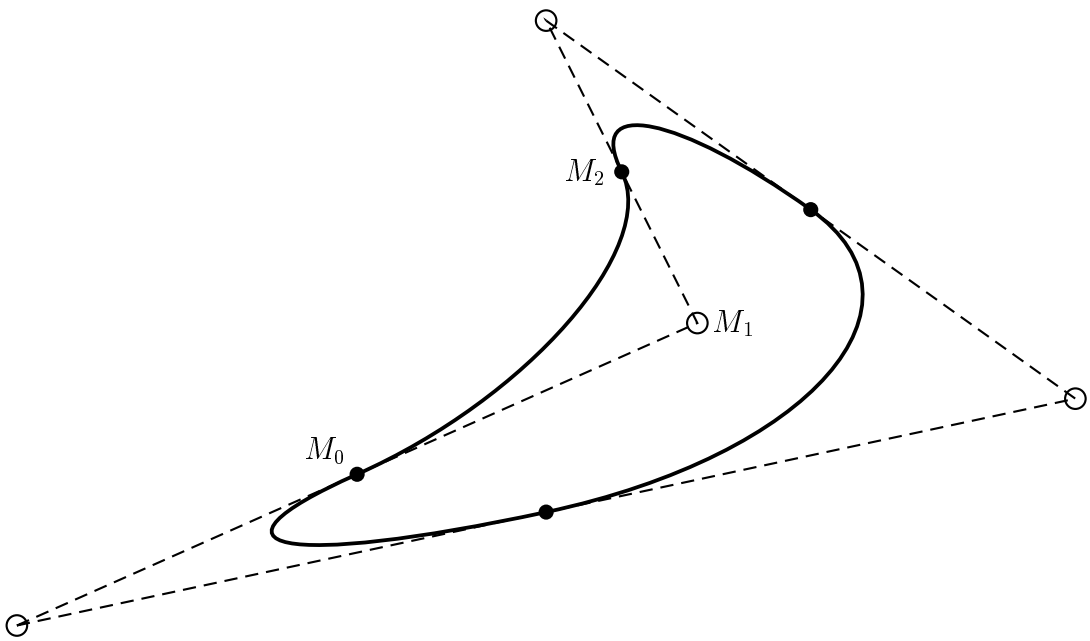
With $k = \dfrac{1}{n}$ and $inc1_x(t)$ which can be defined as:

$$\begin{aligned} inc1_x(0) &= a_x k^2 + b_x k \\ inc1_x(t+k) &= inc1_x(t) + \underbrace{2a_x k^2}_{inc2_x} \end{aligned}$$

In the following pseudocode we'll assume we want 2^n points, thus allowing fixed point arithmetic, which is faster with the use of shift operators in most processors.
So we multiply every value by 2^{2n} (ie shift left by $2n$, written $\ll 2n$), and we have $k = \dfrac{1}{2^n} \ll 2n = 1 \ll n$ and $k^2 = \dfrac{1}{2^{2n}} \ll 2n = 1$.

```
x ← x_0 ≪ 2n
y ← y_0 ≪ 2n
inc1_x ← (b_x ≪ n) + a_x
inc1_y ← (b_y ≪ n) + a_y
inc2_x ← 2a_x
inc2_y ← 2a_y
FOR i ← 0 TO 2^n
    PLOT(x ≫ 2n;y ≫ 2n)
    x ← x + inc1_x
    y ← y + inc1_y
    inc1x ← inc1_x + inc2_x
    inc1y ← inc1_y + inc2_y
NEXT i
```

When you'll try to join two quadratic splines you'll see their major drawback appearing, as said earlier, with the lack of control of derivatives. I've not used them for practical modelling, only in a demonstration (short programs, usually written in assembly, to show the skills of the programmer or the skills of a fellow graphist or musician), but here's the way i controlled them: the example shows the 4 control points (white dots). Then the middle of consecutive control points were computed (black dots), and all 4 splines are defined by three points: two black and one white.



(unimportant) details of the coefficients' calculation

$$\ldots \iff \begin{cases} c_x = x_0 \\ c_y = y_0 \\ a_x + b_x + c_x = x_2 \\ a_y + b_y + c_y = y_2 \\ b_x = \lambda(x_1 - x_0) \\ b_y = \lambda(y_1 - y_0) \\ 2a_x + b_x = \mu(x_2 - x_1) \\ 2a_y + b_y = \mu(y_2 - y_1) \end{cases} \iff \begin{cases} c_x = x_0 \\ c_y = y_0 \\ a_x + b_x = x_2 - x_0 & (1) \\ a_y + b_y = y_2 - y_0 & (2) \\ b_y(x_1 - x_0) = b_x(y_1 - y_0) & (3) \\ (2a_y + b_y)(x_2 - x_1) = (2a_x + b_x)(y_2 - y_1) & (4) \end{cases}$$

Now we modify $(2a_x + b_x)(y_2 - y_1)(y_1 - y_0)$ in two different ways so that only the b_y coefficient appears in each resulting expression (i bet one can be more clever here, but who cares...):

$$\begin{aligned} (2a_x + b_x)(y_2 - y_1)(y_1 - y_0) &= [2(a_x + b_x) - b_x](y_2 - y_1)(y_1 - y_0) \\ &= 2(a_x + b_x)(y_2 - y_1)(y_1 - y_0) - b_x(y_2 - y_1)(y_1 - y_0) \\ &= 2(a_x + b_x)(y_2 - y_1)(y_1 - y_0) - b_y(x_1 - x_0)(y_2 - y_1) && \text{after (3)} \\ &= 2(x_2 - x_0)(y_2 - y_1)(y_1 - y_0) - b_y(x_1 - x_0)(y_2 - y_1) && \text{after (1)} \end{aligned}$$

$$\begin{aligned} (2a_x + b_x)(y_2 - y_1)(y_1 - y_0) &= (2a_y + b_y)(x_2 - x_1)(y_1 - y_0) && \text{after (4)} \\ &= [2(a_y + b_y) - b_y](x_2 - x_1)(y_1 - y_0) \\ &= [2(y_2 - y_0) - b_y](x_2 - x_1)(y_1 - y_0) && \text{after (2)} \\ &= 2(y_2 - y_0)(x_2 - x_1)(y_1 - y_0) - b_y(x_2 - x_1)(y_1 - y_0) \end{aligned}$$

And now by identifying both resulting expressions we find:

$$b_y[(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)] = 2(y_1 - y_0)[(y_2 - y_0)(x_2 - x_1) - (x_2 - x_0)(y_2 - y_1)]$$

$$\iff b_y(x_2y_1 - x_2y_0 - x_1y_1 + x_1y_0 - x_1y_2 + x_1y_1 + x_0y_2 - x_0y_1) = \ldots$$

$$\ldots = 2(y_1 - y_0)(x_2y_2 - x_1y_2 - x_2y_0 + x_1y_0 - x_2y_2 + x_2y_1 + x_0y_2 - x_0y_1)$$

$$\iff b_y(x_2y_1 - x_2y_0 + x_1y_0 - x_1y_2 + x_0y_2 - x_0y_1) = 2(y_1 - y_0)(-x_1y_2 - x_2y_0 + x_1y_0 + x_2y_1 + x_0y_2 - x_0y_1)$$

Both huge expressions in parenthesis are identical, and unless they are equal to zero (which happens when M_0, M_1, M_2 are aligned if i'm not mistaken, but the result below still holds) we can simplify to find:

$$\iff b_y = 2(y_1 - y_0)$$

b_x will be find in a similar way, and then finding a_x and a_y is simple, it can be left as an exercise for the reader as they say in serious books. :)