Practical hints to implement real-time writing on tablets

Implement an algorithm that incrementally plots segments of cubic Hermite splines.

Do not use d3 or another library directly.

d3 cannot draw incrementally, it only can draw the whole stroke. This slows the implementation down quite a lot and leads to the annoying lag between the plotted line and the tip of the stylus.

Do not re-plot the whole canvas while writing

This makes the algorithm slow and introduces lag. Occasionally, a segment may be missing, but this is acceptable.

Do not use cubic B-splines

B-splines are approximating, not interpolating. They do not go through the event positions. Therefore, they do not preserve the authenticity of a signature.

Try to get as much data as possible

Write a fast algorithm. Use the additional events in getCoalescedEvents.

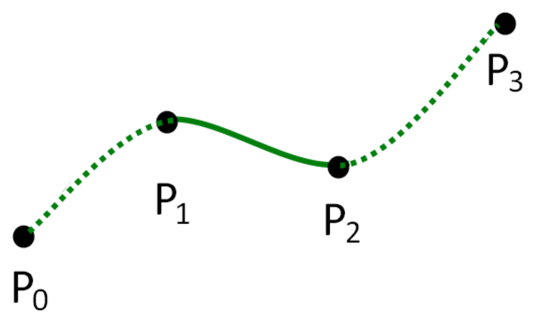


Figure 1: A spline defined by four points

Do a sensible approximation at the start of a stroke

Connect the first two dots by a straight line (P_0 and P_1). Reason: Everyone starts writing slowly, so the length of the first few stroke segments is very short. An edge between the first and the second segment is effectively not visible.

Do a sensible approximation while writing

Do not plot the last spline segment while writing. Nobody will realise. At time t_k, render segment P_{k-2} to P_{k-1}.

If you slow down while writing, the segments become very short, so the (intentional) lag between the end of the rendered line and the tip of the stylus is effectively not visible.

In the picture:

At time 2 draw the straight line P_0 to P_1.

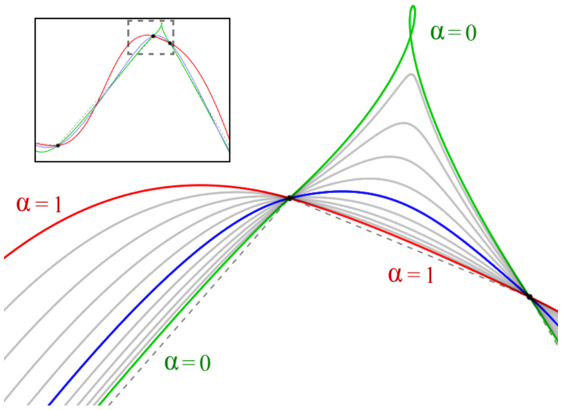
At time 3 draw the spline P_1 to P_2.

Plot the last two segments when drawing has stopped and do again a sensible approximation

If drawing stops at time n, draw the spline segment P_{n-2} to P_{n-1}. Connect P_{n-1} and P_n by a straight line, reason is the same as for the first segment.

Use a Catmull-Rom spline with $\alpha = 0.5$

This spline is proven not to have loops.



Relation between cubic Hermite splines and Bezier curves

Definition of a cubic Hermite spline:

$$S_k(t) \quad = \quad h_{00}(t)P_k + h_{10}(t)M_k + h_{01}(t)P_{k+1} + h_{11}(t)M_{k+1}$$

h are the Hermite basis functions, see table.

P_0 is the starting point

P_1 is the end point

M_0 is the tangent at the start point

M_1 is the tangent at the end point

t is the coordinate scaled to unit interval: $t = \frac{x - x_k}{x_{k+1} - x_k}$

Definition of cubic Bernstein Polynomials

$$B_{k,3}(t) \quad = \quad \binom{3}{k} t^k (1-t)^{3-k} \left[\text{Note: } \binom{3}{k} \text{ are binomial coefficients} \right]$$

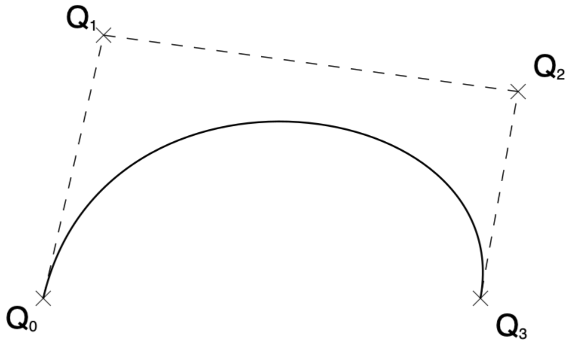


Figure 2: A cubic Bezier curve and its control points.

Hermite Basis functions:

$$
\begin{array}{rrrrrrrr}
 & \text{expanded} & \text{factorised} & \text{Bernstein} & & & & \\
h_{00}(t) = & 2t^3 - 3t^2 + 1 = & (1+2t)(1-t)^2 = & B_0(t) + B_1(t) & h_{00}(0) = & 1 & h_{00}(1) = & 0 \\
h_{10}(t) = & t^3 - 2t^2 + t = & t(1-t)^2 = & \frac{1}{3} \cdot B_1(t) & h_{10}(0) = & 0 & h_{10}(1) = & 0 \\
h_{01}(t) = & -2t^3 + 3t^2 = & t^2(3-2t) = & B_3(t) + B_2(t) & h_{01}(0) = & 0 & h_{01}(1) = & 1 \\
h_{11}(t) = & t^3 - t^2 = & t^2(t-1) = & -\frac{1}{3} \cdot B_2(t) & h_{11}(0) = & 0 & h_{11}(1) = & 0
\end{array}
$$

Cubic Hermite spline re-written in terms of Bernstein polynomials:

$$S_k(t) = B_0(t)P_0 + B_1(t)\left(P_0 + \frac{1}{3}M_0\right) + B_2(t)\left(P_1 - \frac{1}{3}M_1\right) + B_3(t)P_1$$

Definition of a Bezier curve in terms of Bernstein polynomials:

$$
\begin{aligned}
B(t) &= (1-t)^3 Q_0 + 3(1-t)^2 Q_1 + 3(1-t)t^2 Q_2 + t^3 Q_3 \\
&= B_0(t)Q_0 + B_1(t)Q_1 + B_2(t)Q_2 + B_3(t)Q_3
\end{aligned}
$$

Mapping between a cubic Hermite spline segment and a Bezier curve segment:

$$
\begin{aligned}
Q_0 &= P_0 \\
Q_1 &= \left(P_0 + \frac{1}{3}M_0\right) \\
Q_2 &= \left(P_1 - \frac{1}{3}M_1\right) \\
Q_3 &= P_1
\end{aligned}
$$

Mapping formula for Catmull Rom Spline

To determine if a curve segment of the Catmull-Rom curve has a self-intersection, we will convert the polynomial to Bézier form. Let $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ be four consecutive control points of the Catmull-Rom curve with parameter values $0, d_1^\alpha, d_2^\alpha + d_1^\alpha, d_3^\alpha + d_2^\alpha + d_1^\alpha$, where $d_i = |\mathbf{P}_i - \mathbf{P}_{i-1}|$ as shown in Figure 4. The control points of the cubic Bézier curve $\mathbf{B}_j$ ($j \in \{0,1,2,3\}$) representing this polynomial between d_1^α and $d_2^\alpha + d_1^\alpha$, reparameterized to lie in the range $[0,1]$ are then

$$\begin{aligned}
\mathbf{B}_0 &= \mathbf{P}_1 \\
\mathbf{B}_1 &= \frac{d_1^{2\alpha}\mathbf{P}_2 - d_2^{2\alpha}\mathbf{P}_0 + (2d_1^{2\alpha} + 3d_1^\alpha d_2^\alpha + d_2^{2\alpha})\mathbf{P}_1}{3d_1^\alpha(d_1^\alpha + d_2^\alpha)} \\
\mathbf{B}_2 &= \frac{d_3^{2\alpha}\mathbf{P}_1 - d_2^{2\alpha}\mathbf{P}_3 + (2d_3^{2\alpha} + 3d_3^\alpha d_2^\alpha + d_2^{2\alpha})\mathbf{P}_2}{3d_3^\alpha(d_3^\alpha + d_2^\alpha)} \\
\mathbf{B}_3 &= \mathbf{P}_2.
\end{aligned} \tag{2}$$

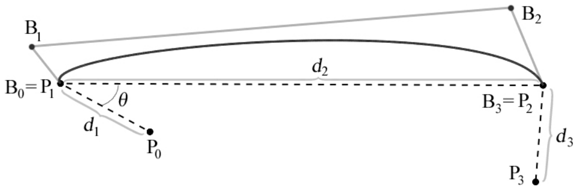


Figure 4: Control points $\mathbf{B}_j$ of the cubic Bézier curve constructed from cubic Catmull-Rom curve segment with control points $\mathbf{P}_i$.

$$xn = P_n$$
$$xn1 = P_{n-1}$$
$$xn2 = P_{n-2}$$
$$xn3 = P_{n-3}$$
$$(x1, y1) = Q_1$$
$$(x2, y2) = Q_2$$

```
var alpha=.5;
var dx=xn-xn1;
var dy=yn-yn1;
var l23_2a = Math.pow(dx * dx + dy * dy, alpha)
var l23_a = Math.sqrt(l23_2a);
dx=xn1-xn2;
dy=yn1-yn2;
var l12_2a = Math.pow(dx * dx + dy * dy, alpha)
var l12_a = Math.sqrt(l12_2a);
dx=xn2-xn3;
dy=yn2-yn3;
var l01_2a = Math.pow(dx * dx + dy * dy, alpha)
var l01_a = Math.sqrt(l01_2a);
x1=xn2;
y1=yn2;
x2=xn1;
y2=yn1;

if (l01_a > 0){
var a = 2 * l01_2a + 3 * l01_a * l12_a + l12_2a;
var n = 3 * l01_a * (l01_a + l12_a);
x1 = (xn2 * a - xn3 * l12_2a + xn1 * l01_2a) / n;
y1 = (yn2 * a - yn3 * l12_2a + yn1 * l01_2a) / n;
}
```

```
if (l23_a > 0){
var b = 2 * l23_2a + 3 * l23_a * l12_a + l12_2a;
var m = 3 * l23_a * (l23_a + l12_a);
x2 = (xn1 * b + xn2 * l23_2a - xn * l12_2a) / m;
y2 = (yn1 * b + yn2 * l23_2a - yn * l12_2a) / m;
}
// html
ctx.moveTo(xn2,yn2);
ctx.bezierCurveTo(x1, y1, x2, y2, xn1, yn1);
// svg (not tested)
M(xn2,yn2)
C(x1, y1, x2, y2, xn1, yn1)
```