Using Cubic Spline Basis Functions for Image Pixel Interpolation

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Some very hairy math involving **Bernstein Polynominal Basis Functions** can greatly ease the understanding and use of Bezier curves and cubic splines. Basis functions also can very much simplify drawing the actual curves. Often, simple table lookups can replace complex cubic calculations. Especially when using splines to do image expansion or interpolation.

Cubic spline fundamentals appear in my **Cubic Spline Library** and include **this tutorial**. In general, a cubic spline is a way to draw a smooth curve that starts at **x0,y0**, ends at **x3,y3** and whose exact shape can be influenced or controlled by intermediate control point pairs **x1,y1** and **x2,y2**. The intermediate points will determine the exit or entry slope and the enthuasiasm of initial and final travel. Four data point pairs thus define the entire curve.

A cubic spline is normally defined as a pair of parametric equations in which **t** usually varies from **0 to 1**...

$$x = at^3 + bt^2 + ct + d$$
$$y = et^3 + ft^2 + gt + h$$

The cubic constants **a** through **h** can be related to the control points by using **this math**. Because **t** changes faster on the "more bent" portions of the curve, directly relating **y** to **x** is nontrivial. Especially since multiple **y** values can result inside a loop or a cusp.

Instead, let's start with an obvious **1=1** and morph it into a bizarre $t + (1-t) = 1$

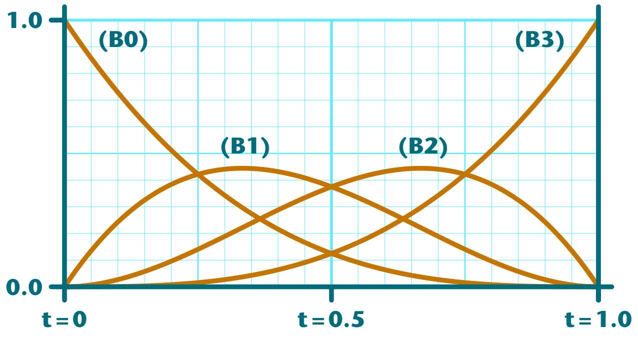
Now, let's cube both sides...

$$t^3 + 3t^2(1-t) + 3t(1-t)^2 + (1-t)^3 = 1$$

or, defining each term as a basis function...

$$B3(t) + B2(t) + B1(t) + B0(t) = 1$$

Let's plot these dudes and see what they look like...



We can see two rather remarkable properties here:

The basis function peaks happen at the spline's control points.

and...

Any (t) value is the sum of the four basis functions evaluated at (t) scaled by their control point values.

Or, in mathspeak...

x(t) = x0*B0(t) + x1*B1(t) + x2*B2(t) + x3*B3(t)
y(t) = y0*B0(t) + y1*B1(t) + y2*B2(t) + y3*B3(t)

Note that the influence points are always at **t = 1/3** and **t = 2/3**, regardless of where they lie in **x-y** space. And that the four unscaled basis point values always add up to one for any allowed value of **t**.

The easiest way to actually plot a cubic spline is with the **curveto** operator in **PostScript**. A fast language independent differental method is **shown here**.

An image interpolation application

Image interpolation can be needed any time you want to make a bitmap larger or smaller. Or any time you are resampling or rectifying or masking or changing geometry. The object is to make the image larger or smaller or distorted without introducing artifacts or looking bad. A very useful tutorial on cubic spline image interpolation **appears here**.

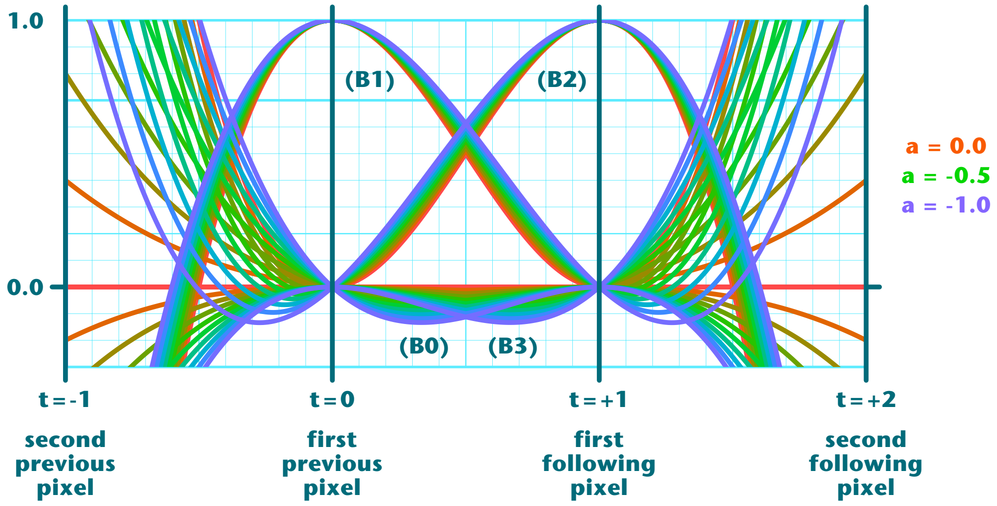
The trick is to take two previous pixels and two post pixels on a scan line and suitably evaluate them to create one or more credible new pixels. These new intermediate results can then be used vertically with four scan lines to create the actual final interpolated pixels.

Doing the obvious of using the four pixels as control points for a cubic spline works. But tends to be blurry because of excessive low pass filtering. We would also like a "knob" that lets us adjust how much sharpening we can get.

Instead, some new **high resolution** basis functions of…

$$B0(t) = at^3 - 2at^2 + at$$
$$B1(t) = (a+2)t^3 - (a+3)t^2 + 1$$
$$B2(t) = -(a+2)t^3 + (2a+3)t^2 - at$$
$$B3(t) = -at^3 + at^2$$

… can be used to advantage as pixel interpolators. Here is how they plot…



Note that we normally only use the region from **t=0** to **t=1**.

If **a=0** , we blur somewhat and use only the adjacent pixel info. This is pretty much the same as the older **bilineal interpolation**.

An **a=-0.5** selection is often best for data to be calculated, such as a medical or astronomical image. And an **a=-1.0** choice is often best for people-viewed results as it includes additional edge enhancement.

The needed basis calculations can be done once and then used as **fast table lookups**. Relating a general spline's **y** to **x** rather than to **t** is **often difficult**. But since the **x** values here will always be **-1**, **0**, **1**, and **2**, our **x(t)** simplifies to…

$$x(t) = -2(2a+1)t^3 + 3(2a+1)t^2 - 2at$$

From which the inverse **t(x)** is easily found as…

$$t(x) = 2(2a+1)x^3 - 3(2a+1)x^2 + 2(a+1)x$$

Underflows or overflows can sometimes happen with sudden changes near max or min intensity. These rare events should be suitably clipped.

Additional support on these concepts can be **found here**.