Finding the Length of a Bezier Cubic Spline and its Subdivisions

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright ©2005 as GuruGram #60.
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Bezier Cubic Splines are an excellent and preferred method to draw the smooth continuous curves often found in typography, CAD/CAM, and graphics in general.

Among their many advantages is a very sparse data set allowing a mere eight values (or four x,y points) to completely define a full and carefully controlled and device independent curve. Many tutorials and examples are now present in our Cubic Spline Library. A brief and useful intro appears here. The fundamental math behind Cubic Splines appears here.

It is often of interest to try and find the exact length of a given cubic spline. First to find the total length itself, and secondarily to be able to subdivide the spline or find an exact point along its length for text positioning or other alignment.

It turns out that an exact closed form solution to the cubic spline length problem is unbearably gruesome as it involves elliptic integrals on top of possible cusp discontinuities and multiple values. Instead…

The exact length of a cubic spline is very difficult to find. Instead, for most uses, a chorded approximation is easier.

Typically, 100 chords will suffice and will be accurate to a fraction of a percent or better.

I’ve put together a set of utility routines as BZLNSUB1.PSL that will rapidly let you find spline lengths. In addition, you can also find the accurate x and y values for any subdivided position along the spline curve.

As with most of our utilities, this is written in raw PostScript, makes optional use of my Gonzo Utilities, and is used to create standard ASCII text files sent on to Acrobat Distiller. Used as a General Purpose Host Based PostScript Interpreter.

A crucial point before we continue…
The "t" parameter is in general nonlinear and tends to change faster along the "more bent" curve portions.

While a nonlinear "t" can be effectively used to find spline lengths, it most assuredly can NOT be used to find exact linear positions ALONG the spline curve.

The workaround is to create two linked and related arrays, one in "t" space and one in accumulated or integrated "s" space that can be interpolated.

Coincidentally, the very last entry in the "s" space accumulated length array will also be the spline length.

Thus, finding total spline lengths is fairly easy. Finding exact linear subdivision points in "s" space is considerably more difficult. But solve the latter and the total length pops out free. Which is the approach we will use here.

What follows is best understood if you have recently reviewed CUBEZMATH.PDF and have a copy of BZLNSUB1.PSL in an open window.

In general, a typical cubic spline is entered as eight data points $x_0,y_0$ up through $x_3,y_3$. With the end points setting the ends of the curve, and the mid points determining the slope and enthusiasm (or tension) of the curve shape.

We can start with a few of the lower level service routines. A first order of business is relating $x_0,y_0$ through $x_3,y_3$ to the A-H parameters of the cubic spline equations themselves...

```
/FindAH {
    /A x3 x2 3 mul x1 3 mul add x0 sub store
    /E y3 y2 3 mul y1 3 mul add y0 sub store
    /B x2 3 mul x1 6 mul sub x0 3 mul add store
    /F y2 3 mul y1 6 mul sub y0 3 mul add store
    /C x1 3 mul x0 3 mul sub store
    /G y1 3 mul y0 3 mul sub store
    /D x0 store /H y0 store
} store
```

Details on where these formulas come from are found in CUBEMATH.PDF.

Next are the "cubeless method" evaluation of x and y data points...
Along with a standard "square root of the sum of the squares" vector calculator to find the length of a chord given its present and previous \( t \) and \( y \) data values...

We need a method to create a \texttt{tlist}, which is simply a numerically ordered array of the \( t \) values. As in \texttt{/tlist [ 0 0.01 0.02 ... 0.99 1.00 ] store}. This is handled by a stock \texttt{/maketlist} routine in \texttt{BZLNSUB1.PSL} that accepts \texttt{numchords} as an input variable. As we’ve seen, a 100 chord approximation will often be acceptable.

\textbf{Now for the Tricky Part…}

You’ll next want to create a \texttt{slist} companion to the \texttt{tlist}. In which the accumulated \( s \) distance appears for any given \( t \) value. For any \( t \) in the array, the new \( x \) and \( y \) values are compared against the previous ones and a new chord is found using \texttt{finds}. These values are then accumulated into the composite \texttt{slist}...

```plaintext
/findx (/ttt exch store A ttt mul B add ttt mul C add ttt
mul D add /curx exch store) store

/findy (/ttt exch store E ttt mul F add ttt mul G add ttt
mul H add /cury exch store) store
```

```plaintext
/finds {curx prevx sub dup mul cury prevy sub dup mul
add sqrt /curs exch store} store
```

```plaintext
/makecurslist {
/prevx 0 store % initialize previous values
/prevy 0 store
/curslist mark 0 % start slist array
0 1 numchords {curtlist exch get % grab current t
/tt exch store
tt findx % find curx and cury
tt findy
finds % find current chord
curs add dup % integrate result; dup for next
/prevx curx store % save new chord end
/prevy cury store
} for
pop ] store % undo final sum, then store
} store
```
Your total spline length is simply the last entry in your slist array...

```
/findsplinelen {curslist dup length 1 sub get
/splinelen exch store} store
```

And that's all you'll really need if your only interest is finding the spline length. Your final high level length-finding code can look something like this...

```
/findbezlen {findAH % calculate cubic coefficients
makecurtlist % create linear t array
makecurslist % create nonlinear s array
findsplinelen % report length as splinelen
} store
```

The BZLNSUB1.PSL utility then goes on to an example or two, noting length values of 31.2185 and 31.3125 and 31.3130 for numchords selections of 10, 100, and 1000. Spline data of 0, 0, 0.01, 0.01, 19.9, 19.9, 20, 20 has a known length of 28.28427 for which a 100 sample length returns 28.28430. Giving us a good accuracy check.

**Secrets of Subdivision**

To find a t for a given s, you work your way through your slist until you find a "bin" that brackets the s you are after. Your entry fraction of that bin can then become your entry fraction on the companion tlist bin...

```
/findtofs { /ss exch store % desired subdivision length
1 1 curslist length 1 sub % scan each s value
{ /ll exch store
   ss curslist ll get
   le (exit) if } for
/ll li 1 sub store
curslist li 1 add get
curslist li get sub
ss curslist li get sub
exch div
curtlist li 1 add get
curtlist li get sub mul
curtlist li get add
} store
```

— 60.4 —
Your high level code to return \( x \) and \( y \) for a normalized 0-1 value for \( s \) is then...

\[
\text{/findxyofs} \{ \text{splinelen mul} \} \quad \% \text{denormalize} \\
\text{findtobs} \quad \% \text{interpolated} \ t \ \text{for the LINEAR} \ s \\
\text{dup findx} \quad \% \text{get} \ x(t) \\
\text{findy} \quad \% \text{get} \ y(t) \\
\text{curx cury} \quad \% \text{and return to stack} \\
\} \ \text{store}
\]

Note that \text{findbezlen} must be run before using this routine. BZLNSUB1.PSL then winds things up with this example of a cubic spline that’s subdivided into twenty constant "s" lengths …

For More Help

Additional info on cubic splines can be found on our Cubic Spline library page. As are many dozens of examples of Bezier cubic spline techniques.

Additional consulting services are available per our Infopack services and on a contract or an hourly basis. Additional GuruGrams are found here.

Further GuruGrams await your ongoing support as a Synergetics Partner.