Approximating a Circle or an Ellipse Using Four Bezier Cubic Splines

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

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.

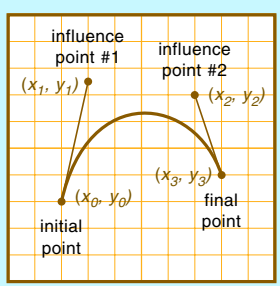
Among their many advantages is a **very sparse data set** allowing a mere **eight** points to **completely** define a full and carefully controlled and device independent curve. Many tutorials and examples are now present in our **Cubic Spline** Library. A brief and useful intro **appears here**.

Cubic splines are exceptionally easy to use in the **PostScript** computer language. But are also generally implementable in most higher level languages and in all but the smallest of bare bones microprocessors. Two obvious things to do with cubic splines are drawing complete **circles** and **ellipses**. It turns out that a mere **four** splines can be used for either task. To an accuracy of well better than one part in one thousand. And thus "good enough" for most graphics and all but the most precise of **CAD/CAM** or optical needs.

Let us begin by excerpting some key **Bezier Cubic Spline properties** from our **HACK62.PDF** tutorial…

Here is a cubic spline shown in its **graph space**…

The **first** influence point sets the direction and the enthusiasm that the spline **leaves** the **initial** point on the curve.



The **second** influence point sets the direction and the enthusiasm that the spline **enters** the **final** point on the curve.

Here is how a cubic spline appears in its **equation space**…

$$x = At^3 + Bt^2 + Ct + D$$
$$y = Et^3 + Ft^2 + Gt + H$$

t (for time) always goes from **zero** at the **initial** point to a **one** at the **final** point.

This is a faster "cube free" form of the **equation space** math…

$$x = (((At) + B)t + C)t + D$$
$$y = (((Et) + F)t + G)t + H$$

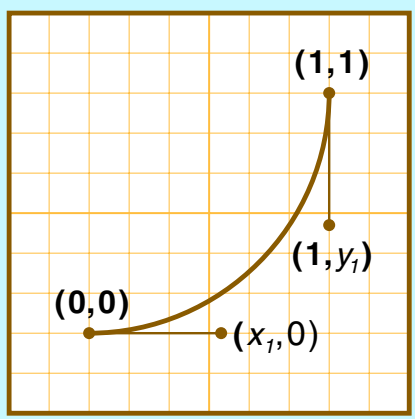
How to get from **graph space** to **equation space**…

$$A = x_3 - 3x_2 + 3x_1 - x_0 \qquad E = y_3 - 3y_2 + 3y_1 - y_0$$
$$B = 3x_2 - 6x_1 + 3x_0 \qquad F = 3y_2 - 6y_1 + 3y_0$$
$$C = 3x_1 - 3x_0 \qquad G = 3y_1 - 3y_0$$
$$D = x_0 \qquad H = y_0$$

How to get from **equation space** to **graph space**…

$$x_0 = D \qquad y_0 = H$$
$$x_1 = D + C/3 \qquad y_1 = H + G/3$$
$$x_2 = D + 2C/3 + B/3 \qquad y_2 = H + 2G/3 + F/3$$
$$x_3 = D + C + B + A \qquad y_3 = H + G + F + E$$

Some of the earlier web derivations of the single **magic number** math needed for a four-spline fit tended towards being complex and obtuse. Actually, the initial solution can be quite simple. Here's a prototype unity normalized spline you can scale and rotate and translate to quad fit any ellipse or circle…



We can immediately see that **x0 = 0** and that **x1**, remains our sought after magic variable. And that **x2 = 1** and **x3 = 1**. By symmetry, solving **x1** also gives us **y1**. Plugging these into our above math quickly gives us…

$$x = (3x_1 - 2)t^3 + (3 - 6x_1)t^2 + (3x_1)t$$

If we want to force a **t = 0.5** at the tangent of **45 degrees**, then…

$$0.707107 = 0.125(3x_1 - 2) + 0.25(3 - 6x_1) + 0.5(3x_1)$$

Which directly and simply leads us to…

The NORMAL 4-spline magic number is 0.55228475. And is equal to the normalized distance along the tangent between an initial point and an influence point.

This gives an exact circle fit every 45 degrees and has a worse case error of less than one part in one thousand and an average error of less than one part in two thousand.

Any error is always POSITIVE and OUTSIDE the circle.

This magic number is also FOUR THIRDS of ONE LESS than the SQUARE ROOT OF TWO.

A one part in one thousand error would be one pixel on a screen circle of 14 inches at 73 DPI. It would be just over **one pixel per inch** on a 1200 DPI printer. Both of these are more than good enough for most people most of the time. On the other hand, if you were machining a three inch cylinder for an engine, the three mil error would probably be wildly unacceptable.

Can We Do Better?

Few people realize that the above usual web derivation is **not** the best you can do. Our first clue is that the error is everywhere positive. And that a "best" solution should have nearly equal positive and negative error regions.

How do you measure the errors? Any point on a true circle has to obey…

$$x^2 + y^2 = r^2$$

Thankfully, your **t** values will be nearly linear with your **degree** values. So, for various **t** values, you find an **x** and a **y** pair. Compare their **actual** circle radius to the **expected** unity value to find your peak and average error deviations.

I've always been a great fan of **Newton's Method**. Otherwise known as "**shake the box**". In which you get near a good answer and then keep making minor changes till you get as close as you care to. These techniques were vital to our **Magic Sinewave** energy efficiency developments.

This obviously leads to backing off on the magic constant so long as the distortion keeps reducing. Lo and behold, at a value nearly **0.000501** lower , we find…

A BEST 4-point spline magic number is 0.551784.

This gives an exact circle fit every 30 degrees.

The peak and average errors are 24 percent lower.

Errors alternate POSITIVE and NEGATIVE along the circle.

This magic number is 0.000501 lower than NORMAL.

Some Code

The **PostScript** computer language gives you all sorts of ways to explore **cubic splines**. By using **Distiller as a PostScript Interpreter**, you can easily export values to other projects as needed. Their **arc** and **arcn** operators let you create splines for any angle, and four spline fits for a circle.

Here is how you report a spline on a **PostScript** path…

```
{== == (moveto\\n) print flush }
{== == (lineto\\n) print flush }
{== == == == == == (curveto\\n) print flush }
{(closepath\\n) print flush} pathforall
```

Here are a pair of ellipse and circle drawing **PostScript** utility procs…

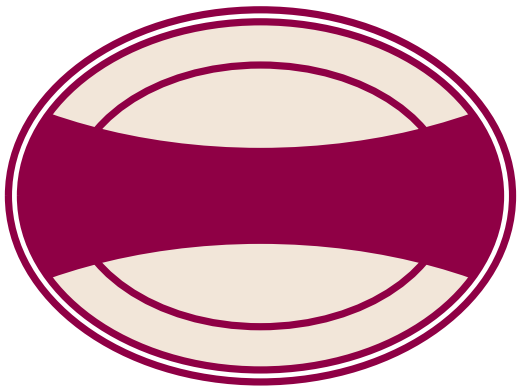
```
/magic 0.55228475 0.00045 sub store  % improved value

/drawellipse {2 div /yrad exch store 2 div /xrad exch store
/xmag xrad magic mul store /ymag yrad magic mul store
xrad  neg 0 moveto xrad neg ymag xmag neg yrad 0 yrad
curveto xmag  yrad xrad ymag xrad 0 curveto xrad ymag
neg  xmag yrad neg 0  yrad neg curveto xmag neg yrad neg
xrad neg ymag neg xrad neg  0 curveto} bind def

/drawcircle {dup drawellipse} bind def
```

To use the ellipse proc, you **pretranslate** so your ellipse is centered on **0,0**. You then enter your **x** diameter and your **y** diameter. Such as **300 200 drawellipse**. For circles, you enter your diameter instead. Such as **250 drawcircle**.

This example might be of use in reconstructing a logo for a **historic interurban railway**…



The actual proc can be found in the **sourcecode** for this **GuruGram**.

Digging Deeper

So, how many splines are needed to approximate a circle to machine shop accuracy? Or, going the other way, how horrendously bad are the two-spline and three-spline circle approximations?

I've done a detailed analysis on this that gives you the full error plots and exact error values. It is available commercially through our **Consulting Services**. Here is a summary of the key results…

The BEST 2-spline magic number is 1.333333. Worst case normalized error deviation is 0.0196725. The average deviation is 0.00869406.

The BEST 3-spline magic number is 0. 0.7698112. Worst case normalized error deviation is 0.00150716. The average deviation is 0.000705631.

The BEST 4-spline magic number is 0.551784. Worst case normalized error deviation is 0.000265718. The average deviation is 0.00012482

The BEST 5-spline magic number is 0.433072. Worst case normalized error deviation is 6.78897e-05. The average deviation is 3.22829e-05.

The BEST 6-spline magic number is 0.3572045. Worst case normalized error deviation is 2.38419e-05. The average deviation is 1.20421e-05.

The BEST 8-spline magic number is 0.2652031. Worst case normalized error deviation is 4.05312e-06. The average deviation is 1.89818e-06

An eight spline circle fit puts you down in the four parts per million worst case and two parts per million average deviation range. Thus, it would seem that **an 8-spline circle fit should be more than adequate for most CAD/CAM or machine shop needs.** Additional splines are probably gross overkill unless approaching optical accuracy is needed.

Once again, the magic number is the normalized distance **along the tangent line** between the entry or exit point and its corresponding influence point.

A Final Detail or two

The behavior of the "**t**" parameter in most any cubic spline is usually subtle and non-obvious. In general, t starts at zero, ends at unity, and changes **faster** along any "**more bent**" portions of the x versus y curve. Surprisingly, on a spline circle approximation, "**t**" **is nearly (but not quite) linear with degrees of arc.**

You can find the exact degrees versus "**t**" relationship by using **inverse trig functions**. Your x and y values represent a unique circle of origin zero.

Here are some of the missing **PostScript** inverse trig procs…

```
/acos {2 copy dup mul exch dup mul sub abs sqrt exch pop
       exch atan} def      % - xside hypotenuse acos -

/asin {2 copy dup mul exch dup mul sub abs sqrt exch pop
       atan} def           % - yside hypotenuse asin -

/trig.acos {1 acos} def    % arcosine from trig value input

/trig.asin {1 asin} def    % arcosine from trig value input
```

Note that the latter two convenience procs **demand** a unity radius.

Normal t versus degree errors are typically a small fraction of a degree. Their only usual consequence is to very slightly skew the average deviation errors.

The expression for the "all error positive" magic number for any number of splines is **4/3[(1 - cosz)/sinz]** degrees per half angle. 4-spline 45 degrees = 0.551784, etc…

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**.