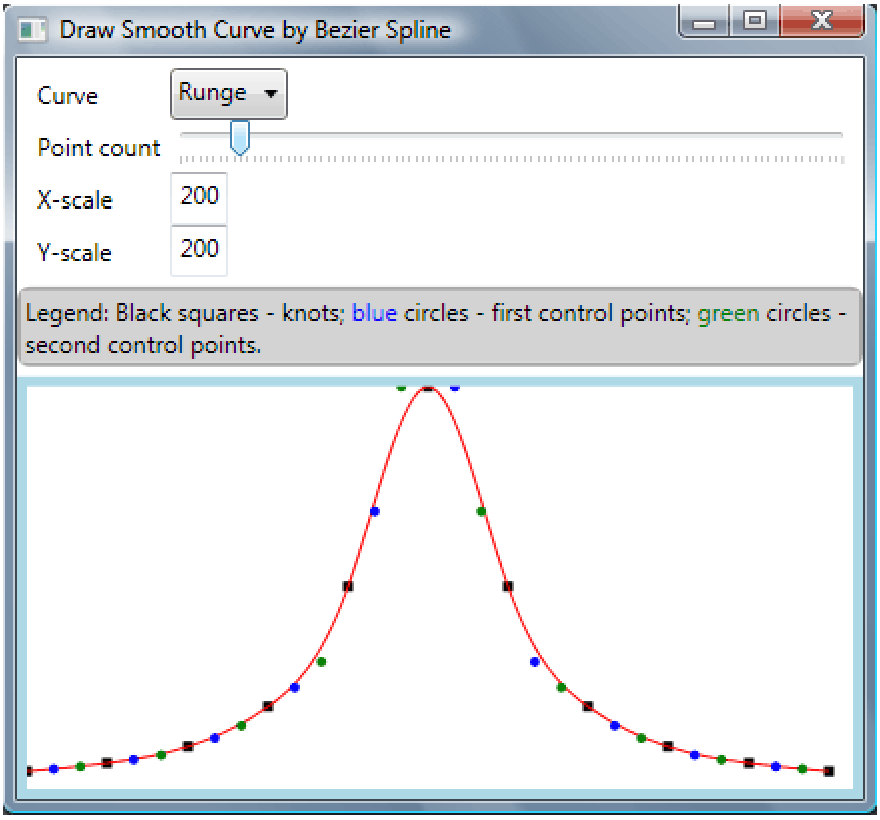
Draw a Smooth Curve through a Set of 2D Points with Bezier Primitives

Oleg V. Polikarpotchkin, Peter Lee, 24 Mar 2009

Calculate piecewise Bezier curve control points to make it a spline

Download source code - 26.12 KB



Introduction

From time to time, I am faced with the question: how to draw a smooth curve through a set of 2D points? It seems strange, but we don't have out of the box primitives to do so. Yes, we can draw a polyline, Bezier polyline, or a piece-wise cardinal spline, but they are all not what is desired: polyline isn't smooth; with Bezier and cardinal spline, we have a headache with additional parameters like Bezier control points or tension. Very often, we don't have any data to evaluate these additional parameters. And, very often, all we need is to draw the curve which passes through the points given and is smooth enough. Let's try to find a solution.

We have interpolation methods at hand. The cubic spline variations, for example, give satisfactory results in most cases. We could use it and draw the result of the interpolation, but there are some nasty drawbacks:

1. Cubic spline is a cubic polynomial, but Win32, .NET Forms, or WPF do not provide methods to draw a piecewise cubic polynomial curve. So, to draw that spline, we have to invent an effective algorithm to approximate a cubic polynomial with polyline, which isn't trivial (word “effective” is a point!).
2. Cubic spline algorithms are usually designed to deal with functions in the Cartesian coordinates, y=f(x). This implies that the function is single valued; it isn't always appropriate.

On the other hand, the platform provides us with a suite of Bezier curve drawing methods. Bezier curve is a special representation of a cubic polynomial expressed in the parametric form (so it isn't the subject of single valued function restriction). Bezier curves start and end with two points often named “knots”; the form of the curve is controlled by two more points known as “control points”.

Bezier spline is a sequence of individual Bezier curves joined to form a whole curve. The trick to making it a spline is to calculate control points in such a way that the whole spline curve has two continuous derivatives.

I spent some time Googling for a code in any C-like language for a Bezier spline, but couldn't found any cool, ready-to-use code.

Here, we'll deal with open-ended curves, but the same approach could be applied to the closed curves.

Bezier Curve Representation

A Bezier curve on a single interval is expressed as:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3 \ldots (1)$$

where `t` is in `[0,1]`, and

1. P_0 – first knot point
2. P_1 – first control point (close to `P0`)
3. P_2 – second control point (close to `P3`)
4. P_3 – second knot point

The first derivative of (1) is:

$$B'(t) = -3(1-t)^2 P_0 + 3(3t^2 - 4t + 1)P_1 + 3(2t - 3t^2)P_2 + 3t^2 P_3$$

The second derivative of (1) is:

$$B''(t) = 6(1-t)P_0 + 3(6t-4)P_1 + 3(2-6t)P_2 + 6tP_3$$

Single Segment

If we have just two knot points, our "smooth" Bezier curve should be a straight line, i.e. in (1) the coefficients in the members with the power 2 and 3 should be zero. It's easy to deduce that its control points should be calculated as:

$$3P_1 = 2P_0 + P_3 P_2 = 2P_1 - P_0$$

Multiple Segments

This is the case where we have more than two points. One more time: to make a sequence of individual Bezier curves to be a spline, we should calculate Bezier control points so that the spline curve has two continuous derivatives at knot points.

Considering a set of piecewise Bezier curves with `n+1` points and `n` subintervals, the `(i-1)`-th curve should connect to the `i`-th one. Now we will denote the points as follows:

1. P_i – `i`th knot point (`i=1,..,n`)
2. $P1_i$ – first control point close to P_i
3. $P2_i$ – second control point close to P_i

At the i^{th} subinterval, the Bezier curve will be:

$$B_i(t) = (1-t)^3 P_{i-1} + 3(1-t)^2 t P1_i + 3(1-t)t^2 P2_i + t^3 P_i \ldots (i = 1,..,n)$$

The first derivative at the i^{th} subinterval is:

$$B_i'(t) = -3(1-t)^2 P_i - 1 + 3(3t^2 - 4t + 1)P1_i + 3(2t - 3t^2)P2_i + 3t^2 P_i; \ldots (i = 1,..,n)$$

The first derivative continuity condition

$$B_{i-1}'(1) = B_i'(0)$$

gives:

$$P1_i + P2_{i-1} = 2Pi - 1; \ldots (i = 2,..,n)(2)$$

The second derivative at the i^{th} subinterval is:

$$B_i''(t) = 6(1-t)P_{i-1} + 6(3t-2)P1_i + 6(1-3t)P2_i + 6tP_i; \ldots (i = 1,..,n)$$

The second derivative continuity condition $B''_{i-1}(1)=B''_i(0)$ gives:

$$P1_{i-1} + 2P1_i = P2_i + 2P2_{i-1}; \ldots (i = 2,..,n)(3)$$

$P1_{i-1}+2P1_i=P2_i+2P2_{i-1}$; (i=2,..,n) (3)

Then, as always with splines, we'll add two more conditions at the ends of the total interval. These will be the “natural conditions” `B’’<sub>1</sub>(0) = 0` and `B’’<sub>n</sub>(1) = 0`.

$$2P1_1 - P2_1 = P_0(4)2P2_n - P1_n = P_n \ldots (5)$$

$2P1_1$-$P2_1$ = P_0 (4) $2P2_n$-$P1_n$ = P_n (5)

Now, we have `2n` conditions (2-5) for `n` control points `P1` and `n` control points `P2`. Excluding `P2`, we'll have `n` equations for `n` control points `P1`:

$$\begin{aligned} 2P1_1 + P1_2 &= P_0 + 2P_1 P1_1 + 4P1_2 + P1_3 = 4P_1 + 2P_2 \\ P1_{i-1} + 4P1_i + P1_{i+1} &= 4P_{i-1} + 2P_i \ldots (6) \\ P1_{n-2} + 4P1_{n-1} + P1_n &= 4P_{n-2} + 2P_{n-1} 2P1_{n-1} + 7P1_n = 8P_{n-1} + P_n \end{aligned}$$

System (6) is the tridiagonal one with the diagonal dominance, hence we can solve it by simple variable elimination without any tricks.

When `P1` is found, `P2` could be calculated from (2) and (5).

The Code

Hide Shrink Copy Code

```
/// <summary>
/// Bezier Spline methods
/// </summary>
public static class BezierSpline
{
    /// <summary>
    /// Get open-ended Bezier Spline Control Points.
    /// </summary>
    /// <param name="knots">Input Knot Bezier spline points.</param>
    /// <param name="firstControlPoints">Output First Control points
    /// array of knots.Length - 1 length.</param>
    /// <param name="secondControlPoints">Output Second Control points
    /// array of knots.Length - 1 length.</param>
    /// <exception cref="ArgumentNullException"><paramref name="knots"/>
    /// parameter must be not null.</exception>
    /// <exception cref="ArgumentException"><paramref name="knots"/>
    /// array must contain at least two points.</exception>
    public static void GetCurveControlPoints(Point[] knots,
        out Point[] firstControlPoints, out Point[] secondControlPoints)
```

```
{
    if (knots == null)
        throw new ArgumentNullException("knots");
    int n = knots.Length - 1;
    if (n < 1)
        throw new ArgumentException
        ("At least two knot points required", "knots");
    if (n == 1)
    { // Special case: Bezier curve should be a straight line.
        firstControlPoints = new Point[1];
        // 3P1 = 2P0 + P3
        firstControlPoints[0].X = (2 * knots[0].X + knots[1].X) / 3;
        firstControlPoints[0].Y = (2 * knots[0].Y + knots[1].Y) / 3;

        secondControlPoints = new Point[1];
        // P2 = 2P1 – P0
        secondControlPoints[0].X = 2 *
            firstControlPoints[0].X - knots[0].X;
        secondControlPoints[0].Y = 2 *
            firstControlPoints[0].Y - knots[0].Y;
        return;
    }

    // Calculate first Bezier control points
    // Right hand side vector
    double[] rhs = new double[n];

    // Set right hand side X values
    for (int i = 1; i < n - 1; ++i)
        rhs[i] = 4 * knots[i].X + 2 * knots[i + 1].X;
    rhs[0] = knots[0].X + 2 * knots[1].X;
    rhs[n - 1] = (8 * knots[n - 1].X + knots[n].X) / 2.0;
    // Get first control points X-values
    double[] x = GetFirstControlPoints(rhs);

    // Set right hand side Y values
    for (int i = 1; i < n - 1; ++i)
        rhs[i] = 4 * knots[i].Y + 2 * knots[i + 1].Y;
    rhs[0] = knots[0].Y + 2 * knots[1].Y;
    rhs[n - 1] = (8 * knots[n - 1].Y + knots[n].Y) / 2.0;
    // Get first control points Y-values
    double[] y = GetFirstControlPoints(rhs);

    // Fill output arrays.
    firstControlPoints = new Point[n];
    secondControlPoints = new Point[n];
    for (int i = 0; i < n; ++i)
    {
        // First control point
        firstControlPoints[i] = new Point(x[i], y[i]);
        // Second control point
        if (i < n - 1)
            secondControlPoints[i] = new Point(2 * knots
                [i + 1].X - x[i + 1], 2 *
                knots[i + 1].Y - y[i + 1]);
        else
            secondControlPoints[i] = new Point((knots
                [n].X + x[n - 1]) / 2,
                (knots[n].Y + y[n - 1]) / 2);
    }
}

/// <summary>
/// Solves a tridiagonal system for one of coordinates (x or y)
/// of first Bezier control points.
/// </summary>
/// <param name="rhs">Right hand side vector.</param>
/// <returns>Solution vector.</returns>
private static double[] GetFirstControlPoints(double[] rhs)
{
    int n = rhs.Length;
    double[] x = new double[n]; // Solution vector.
    double[] tmp = new double[n]; // Temp workspace.
```

```
        double b = 2.0;
        x[0] = rhs[0] / b;
        for (int i = 1; i < n; i++) // Decomposition and forward substitution.
        {
            tmp[i] = 1 / b;
            b = (i < n - 1 ? 4.0 : 3.5) - tmp[i];
            x[i] = (rhs[i] - x[i - 1]) / b;
        }
        for (int i = 1; i < n; i++)
            x[n - i - 1] -= tmp[n - i] * x[n - i]; // Backsubstitution.

        return x;
    }
}
```

Although I compiled this code in C# 3.0, I don't see why it can't be used without any modification in C# 2.0, and even in C# 1.0 if you remove the keyword "`static`" from the class declaration.

Note that the code uses the `System.Windows.Point` structure, so it is intended for use with Windows Presentation Foundation, and the `using System.Windows;` directive is required. But all you should do to use it with Windows Forms is to replace the `System.Windows.Point` type with `System.Drawing.PointF`. Equally, it is straightforward to convert this code to C/C++, if desired.

The Sample

The sample supplied with this article is a Visual Studio 2008 solution targeted at .NET 3.5. It contains a WPF Windows Application project designed to demonstrate some curves drawn with the Bezier spline above. You can select one of the curves from the combo box at the top of the window, experiment with the point counts, and set appropriate XY scales. You can even add your own curve, but this requires coding as follows:

1. Add your curve name to the `CurveNames` enum.
2. Add your curve implementation to the *Curves* region.
3. Add a call to your curve in the `OnRender` override.

In the sample, I use `Path` elements on the custom `Canvas` to render the curve, but in a real application, you would probably use a more effective approach like visual layer rendering.

History

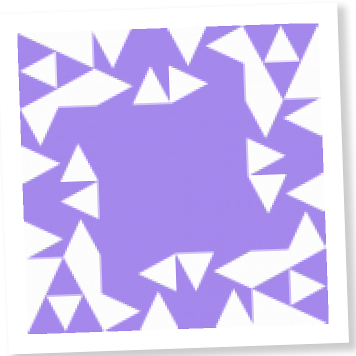
- 18th December, 2008: Initial post
- 23rd March, 2009: Second article revision with the following corrections and additions:
    1. The most important is the bug fix. This bug as well as its correction was found by Peter Lee. A lot of thanks to him! This bug produced visually distinguishable behavior in the case of small knot points count.
    2. `GetCurveControlPoints` now throw exceptions if invalid parameter is passed.
    3. Added special handling of the case where knots array has just two points
    4. Unit tests added.

License

Share

TWITTER FACEBOOK

About the Authors

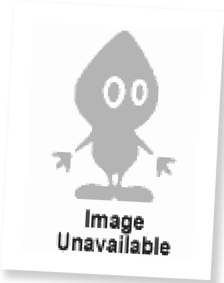


Oleg V. Polikarpotchkin

Team Leader

Russian Federation

No Biography provided



Peter Lee

United States

No Biography provided

You may also be interested in...

A Solution Blueprint for DevOps

REST API or SOAP Testing Automation with ZeroCode JSON-Based Open Source Test Framework

Draw Closed Smooth Curve with Bezier Spline

Creating Web API in ASP.NET Core 2.0

Bezier Curves Made Simple

Interpolate 2D points, usign Bezier curves in WPF

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

Great article
Member 12169498 **26-Nov-15 12:52**

Adding Start and End-Tangent
Member 10077390 **15-Oct-15 0:28**

A brief clarification
a1jrj **1-Dec-12 2:09**

I don't know how these formula can become to that codes....?
Jeffrey6394 **17-Oct-12 0:27**

Cannot see equation
Alexandr.Pi **21-Jun-12 4:34**

Re: Cannot see equation
Peter Lee 22-Jun-12 22:09

Problem with drawing Bezier spline
yamelkaya **14-Jun-12 23:43**

Your HTML is showing
hairy_hats **16-Mar-12 5:41**

how can i generate different number of Points from constant number of knots
engkhaledd **28-Feb-12 11:49**

My vote of 5
Member 8204785 **18-Jan-12 4:35**

Great Job!
Christopher Drake **19-Dec-11 8:59**

My vote of 5
BlazeCell **1-Dec-11 17:59**

Blocks of lines appearing while drawing the curve.
deeptigp **22-Oct-11 8:16**

My vote of 5
Susan Spencer **11-Jun-11 9:34**

Smoth curves for signature
GauravKP **21-Feb-11 2:24**

Use in graph sketching...
xian123 **23-Oct-10 8:21**

tridiagonal system
k_lin **23-May-10 11:27**

More than one control points
dineshreddy.cs **9-Mar-10 0:35**

Curl between 2 knots
Jean-LucM **30-Sep-09 6:09**

Re: Curl between 2 knots

Oleg V. Polikarpotchkin 1-Oct-09 20:41

Re: Curl between 2 knots
Jean-LucM 2-Oct-09 6:43

Re: Curl between 2 knots
Oleg V. Polikarpotchkin 2-Oct-09 19:40

Re: Curl between 2 knots
Jean-LucM 3-Oct-09 3:34

Re: Curl between 2 knots
Oleg V. Polikarpotchkin 3-Oct-09 20:10

Re: Curl between 2 knots
Jean-LucM 6-Oct-09 6:59

Refresh 1 2 3 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.