Piecewise Linear Approximation of Bézier Curves

Kaspar Fischer

October 16, 2000

Abstract

In computer graphics one often needs to convert a given Bézier curve to a polygon (i.e., to a sequence of connected line-segments). These notes explain one way, invented and copyright by Roger Willcocks [2], on how to elegantly compute such piecewise linear approximations of Bézier curves.

1 Introduction

In computer graphics one is quite often confronted with the rendering of two-dimensional cubic Bézier curves. For this and other purposes we would like to convert a given Bézier curve to a polygon, that is, to a series of connected line-segments as illustrated in Figure 1.

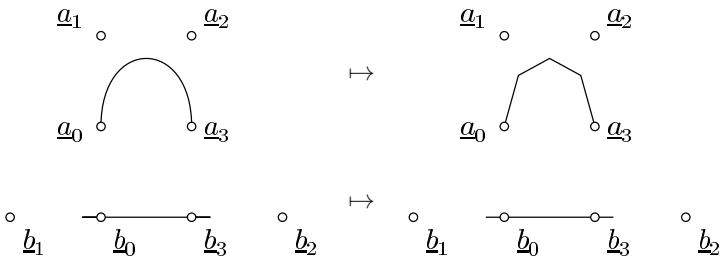

Figure 1: Two curves (left) and their approximations (right).

So assume that we are given a curve $\underline{b}(t)$ with start-point $\underline{b}_0$, the two control-points $\underline{b}_1, \underline{b}_2$, and end-point $\underline{b}_3$:

$$\underline{b}(t) = (1-t)^3\underline{b}_0 + 3\,(1-t)^2 t\underline{b}_1 + 3\,(1-t)\,t^2\underline{b}_2 + t^3\underline{b}_3.$$

There are several possibilities to find polygonal approximations. We could, for instance, sample the curve at some points t_i $(i = 1, \ldots, n)$ in parameter-space. But it's not clear how to choose the values t_i (and the number n) such that flat parts of the curve are sampled with low density and strongly bent parts are approximated with a larger number of line-segments.—It seems best to use a method based on "recursive subdivision."

2 Recursive Subdivision

Recursive subdivision, called the *de Casteljau construction* in this context, is a way to "split" a Bézier curve $\underline{b}(t)$ into two smaller Bézier curves, a left one $\underline{l}(t)$ and a right one $\underline{r}(t)$. The scheme works as illustrated in Figure 2:

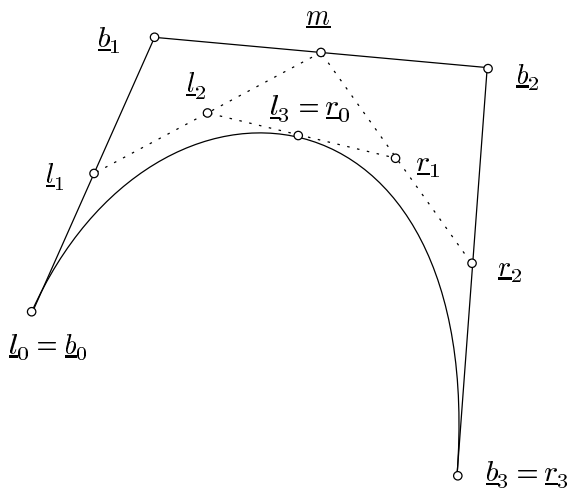

Figure 2: The de Castejau construction.

As indicated in the figure, we calculate the mid-point $\underline{m}$ between $\underline{b}_1$ and $\underline{b}_2$ and proceed likewise for $\underline{l}_1$ and $\underline{r}_2$, $\underline{l}_2$ and $\underline{r}_1$ and finally $\underline{l}_3$. Then, the Bézier curve through points $\underline{l}_0$ and $\underline{l}_3$ with control-points $\underline{l}_1$ and $\underline{l}_2$ coincides with the first part of the original Bézier curve. Similarly, the curve through points $\underline{r}_0$ and $\underline{r}_3$ with control-points $\underline{r}_1$ and $\underline{r}_2$ exactly matches the right part of the orginal curve. To see that this is correct, we can compare the polynomial $\underline{b}(t)$ (the original curve) with the polynomial $\underline{l}(t)$ of the left curve: If $\underline{b}(t)$ equals $\underline{l}(2t)$, the two curves are identical. In Maple we can easily verify the identity in question; here's the test for the x-coordinates:

```
> bezier:= (b,t) -> (1-t)^3*b[0] + 3*t*(1-t)^2*b[1] +
    3*t^2*(1-t)*b[2] + t^3*b[3]:
> m:= (b[1]+b[2])/2:
> l[0]:= b[0]: r[3]:= b[3]:
> l[1]:= (b[0]+b[1])/2:
> r[2]:= (b[2]+b[3])/2:
> l[2]:= (l[1]+m)/2:
> r[1]:= (m+r[2])/2:
> l[3]:= (l[2]+r[1])/2: r[0]:= l[3]:
> expand(bezier(b,t)-bezier(l,2*t));
```

0

Thus the idea to approach our problem is to recursively subdivide the orginal curve $\underline{b}(t)$ until we obtain "sub-curves" which are "sufficiently" flat to be approximated by straight line-segments. In pseudo-code this looks as follows:

```
void flattenCurve(Curve c) {
  if (isSufficientlyFlat(c))
    output(c); /* output as line-segment */
  else {
    Curve l,r;
    subdivide(c,l,r); /* split c into curves l and r */
    flattenCurve(l); /* enumerate left curve */
    flattenCurve(r); /* enumerate right curve */
  }
}
```

3 Is this Curve Flat?

The question remains how to check whether a given curve is "sufficiently" flat. The following approach, both in theory and implementation, is due to Roger Willcocks, author of the RoPS PostScipt interpreter[1]. The method is copyright by Roger Willcocks, and I am using it here with his generous permission [2].

We will measure the flatness of a curve by means of a positive number: Curves with flatness zero will be considered totally flat, while higher and higher values more and more strongly suggest that the curve be further subdivided. Thus, the idea is that the user specifies a tolerance tol such that no curve with flatness greater than tol is approximated by a straight line but is subdivided further instead. Notice that since we are only interested in rejecting non-flat curves, it isn't a problem if we mistakenly continue the subdivision process for a curve which is already flat enough.

More precisely, we define the flatness of a curve $\underline{b}(t)$ to be the number

$$f = \max_{0 \le t \le 1} d(t) \qquad \text{where} \qquad d(t) = \|\underline{b}(t) - \underline{l}(t)\|.$$

Here, $\underline{l}(t)$ is the line-segment $\underline{l}(t) = (1-t)\,\underline{b}_0 + t\underline{b}_3$ from the curve's start- to its end-point. Figure 3 illustrates this definition.

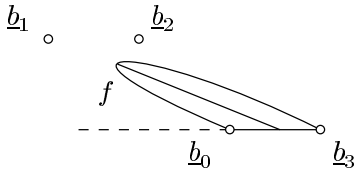

Figure 3: Definition of the flatness f.

The line $\underline{l}(t)$ can be represented as a Bézier curve with start-point $\underline{b}_0$, end-point $\underline{b}_3$, and control-points $(2\underline{b}_0 + \underline{b}_3)/3$ and $(\underline{b}_0 + 2\underline{b}_3)/3$, respectively:

$$\underline{l}(t) = (1-t)^3\underline{b}_0 + (1-t)^2 t\,[2\underline{b}_0 + \underline{b}_3] + (1-t)\,t^2\,[\underline{b}_0 + 2\underline{b}_3] + t^3\underline{b}_3$$

You can easily verify that this curve has indeed constant velocity by computing $\underline{l}'(t)$ which is $\underline{b}_3 - \underline{b}_0$, independent of t. (Or just multiply out the expression

[1] See `http://www.rops.org/`

for $\underline{l}'(t)$ and collect terms—you'll end up with $\underline{l}(t) = (1-t)\,\underline{b}_0 + t\underline{b}_3$.) Continuing, we have

$$\begin{aligned}\underline{b}(t) - \underline{l}(t) &= (1-t)^2 t\,(3\underline{b}_1 - 2\underline{b}_0 - \underline{b}_3) + (1-t)\,t^2\,(3\underline{b}_2 - \underline{b}_0 - 2\underline{b}_3) \\ &= (1-t)\,t\,\left((1-t)\,\underline{u} + t\,\underline{v}\right),\end{aligned}$$

for $\underline{u} = 3\underline{b}_1 - 2\underline{b}_0 - \underline{b}_3$ and $\underline{v} = 3\underline{b}_2 - \underline{b}_0 - 2\underline{b}_3$. Hence

$$\begin{aligned}d^2(t) &= \|\underline{b}(t) - \underline{l}(t)\|^2 \\ &= (1-t)^2 t^2\,\left(\left((1-t)\,u_x + t v_x\right)^2 + \left((1-t)\,u_y + t v_y\right)^2\right)\end{aligned}$$

Since $\max_{0\le t\le 1}(1-t)^2 t^2 = 1/16$ and $\max_{0\le t\le 1}[(1-t)\,a + tb] = \max\{a, b\}$ for any two constants a, b, we finally obtain

$$f^2 = \max_{0\le t\le 1} d^2(t) \le 1/16\,\left(\max\{u_x^2, v_x^2\} + \max\{u_y^2, v_y^2\}\right).$$

(As Roger Willcocks notes, one could easily derive a better bound, but the above can be speedily computed.) In pseudo-code this looks as follows:

```
bool isSufficentlyFlat(Curve c) {
  double ux = 3.0*c.b1_x - 2.0*c.b0_x - c.b3_x; ux *= ux;
  double uy = 3.0*c.b1_y - 2.0*c.b0_y - c.b3_y; uy *= uy;
  double vx = 3.0*c.b2_x - 2.0*c.b3_x - c.b0_x; vx *= vx;
  double vy = 3.0*c.b2_y - 2.0*c.b3_y - c.b0_y; vy *= vy;
  if (ux < vx) ux = vx;
  if (uy < vy) uy = vy;
  return (ux+uy <= tolerance); /* tolerance is 16*tol^2 */
}
```

The test is quite efficient and seems to be very robust in practice. Even for degenerated curves like the curve $\underline{b}$ in Figure 1, the test works as expected: Although this curve is a line (which one would normally consider "flat"), it should *not* (and will not) be considered flat here since we need to subdivide it further to find the start- and end-point of the described line.

Also, our definition of flatness quite nicely matches the one used in Adobe's PostScript language [1] (page 669, operator setflat *flatness*): "Flatness is the error tolerance of this approximation; it is the maximum allowable distance of any point of the approximation from the corresponding point on the true curve, measured in output device pixels." So our number tol has the same semantics as the argument *flatness* to the operator setflat.

References

[1] Adobe Systems Incorporated. *PostScript Language Reference.* Addison-Wesley, Reading, MA, USA, third edition, 1999.

[2] R. Willcocks. September 2000. Personal communication to K. Fischer.