Drawing Parametric Curves Using Chebyshev Polynomials

John Buchanan
Alain Fournier

Department of Computer Science
University of British Columbia
Vancouver, British Columbia
V6T 1W5
{buchanan | fournier}@cs.ubc.ca

Abstract

Polynomial parametric curves are powerful and popular modeling tools in Computer Graphics and Computer Aided Design. There are two requirements that are placed on techniques for displaying these curves. In interactive applications, such as drawing and design, the need is for a fast display. In non interactive applications, such as typesetting, the need is for accuracy (or at least the appearance of accuracy). Most techniques address one or the other of these conflicting requirements.

We propose and demonstrate the use of *Chebyshev basis functions* for an adaptive curve drawing method which is both fast and accurate. The use of Chebyshev polynomials provide us with an inexpensive linearity measure which is useful in an recursive algorithm. Further applications of this approach include efficient boxing and the generation of smooth filtered curves.

Résumé

Les courbes paramètriques polynômiales constituent un outil puissant et populaire pour la modèlisation en infographie et en conception assistée par ordinateur. Les techniques d'affichage pour ces courbes doivent répondre à deux exigences. Dans les applications interactives, telles que le dessin et le design, l'affichage doit être très rapide. Dans les applications non-interactives, telles que la typographie digitale, la précision (en tout cas l'apparence de la précision) est primordiale. La plupart des techniques sont concernées avec l'une ou l'autre de ces exigences.

Nous proposons et nous illustrons l'utilisation des polynômes de Tchebychev comme bases pour une technique adaptive de tracé qui est à la fois rapide est précise. L'utilisation des polynômes de Tchebychev nous fournit une mesure de linéarité économique très utile dans le cadre d'une méthode récursive. D'autres applications de la meme approche incluent une mise en boite efficace et la génèration de courbes lissées et filtrées.

CR Categories: I.3.3 [**Computer Graphics**]: Picture/Image Generation I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms: algorithms.

Additional Keywords and Phrases: Chebyshev polynomials, Linearity criteria, Adaptive curve drawing.

1. Introduction

Polynomial parametric curves are powerful and popular modeling tools in Computer Graphics and Computer Aided Design. A combination of hardware and algorithmic advances have made it possible to draw hundreds or thousands of these curves in real time on modern graphics workstations. There are two requirements that are placed on techniques for displaying these curves. In interactive applications the need is for a fast display, and this can be achieved by drawing long straight line segments, sacrificing the smoothness of the original curve. In non-interactive applications the need is for accuracy. It is interesting to note that what matters is usually the appearance of accuracy. For example a curve with a visible discontinuity is normally perceived as worse than a smooth curve, even though by some metric the curve with the discontinuity might be better than the smooth curve.

Most curve drawing techniques address one or the other of these conflicting requirements. We propose here a technique which is adaptive but fast, and has the advantage of offering a good "quality" criterion as part of the algorithm.

1.1. Current Methods

Since most of the display systems we work with currently are capable of drawing straight lines and setting pixels, the task of drawing a parametric curve can be reduced to that of either drawing a straight line or setting pixels. So an obvious strategy is to subdivide a curve until it can be satisfactorily represented by a succession of straight line segments. There are many methods for subdividing parametric curves but the method of choice has been to perform a uniform subdivision of the curve in the parameter t. The curve is given as:

$$\mathbf{Q}(t) = (X(t), Y(t))$$

where $X(t)$ and $Y(t)$ are polynomials in t. Given a

$\delta_i = t_{i+1} - t_i$, one calculates the coordinates $\mathbf{Q}(t_i)$ and $\mathbf{Q}(t_{i+1})$ and draw the line segment which connects them. The full cost of directly evaluating the curve polynomials can be avoided by using forward differencing (see Foley & van Dam [Fole82] for a description of the technique in this context). Since the forward differencing computations can be put in the form of a multiplication by a 4x4 matrix in the case of cubic curves, in some graphics systems hardware is used to speed up this process. A major drawback of this method is that there is no obvious good heuristic for determining a correct value of δ. The selection of a small δ can result in excessive computation, the selection of a large δ can result in a curve with a polygonal look (*ie* obvious discontinuities) and large "missing parts".

Recursive subdivision is an old numerical method, and Catmull [Catm75] used it in the context of the display of Bézier surfaces. Many other followed, and to mention a still popular method Lane, Carpenter, Whitted and Blinn [Lane80] showed how a curve could be recursively subdivided until each section of the curve could be adequately approximated by a straight line segment. In their method they used the Bernstein polynomial basis and used the convex hull property of the Bézier curves to produce a measure of 'straightness'. Given a Bézier curve defined by the control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$, the straight line segment which approximates this curve is the line from $\mathbf{P}_0$ to $\mathbf{P}_3$ and the error measure is defined by whichever of the points $\mathbf{P}_1$, $\mathbf{P}_2$ is furthest from the approximating line. The cost of calculating this error measure at each level of subdivision is quite high.

Adaptive forward differencing was a method introduced by Lien, Shantz, and Pratt [Prat87 , Chan88 , Rocc89] which attempted to bridge the gap between recursive subdivision and uniform subdivision. In this scheme the Euclidean distance between the current point and the next point is evaluated, if this distance is greater than a pixel the step size is halved and if the distance is smaller than a pixel it is doubled. The curves which are generated using this method can be very accurate but the method suffers from serious drawbacks. First the unwary implementer may end up with an infinite loop as the method doubles and halves the next parametric step in an effort to find a correct step, since the distance is not linearly related to the parametric step. A simple solution to this is to check for the undershoot first, then check for the overshoot, and accept the resulting step. This ensures that a step is always taken. A second and more serious flaw is the possibility of omitting part of the curve, this can occur if a large initial value is selected for δ. In Figure 1 we see a situation where a selection of δ=1 will result in a single pixel being rendered, since the two end points of the curves are geometrically the same. Cases like this will always come up with a method which uses only a local, as opposed to a global, criterion for the span considered.

1.2. Geometric vs Parametric Straightness

There can be two criteria when evaluating the straightness of a parametric curve. The first one, which can be called *geometric straightness*, is related to the distance between the curve and some straight line segment approximation. This would be ideally something like the minimum largest distance over the whole curve, or some value bounding it. The other one, *parametric straightness*, is related to the distance between a point on the straight line segment and the point on the curve with the same parameter. Here again one might be interested in a minmax measure. It of course assumes a parametrization of the line segment, and different parametrizations will give different results. One assumes here a uniform (*ie* linear) parametrization for the line segment. It is clear that the parametric straightness "contains" the geometric straightness, since the distance in the former is always greater than or equal to the distance in the latter. In this respect the distinction is similar to the difference between geometric and parametric continuity (where the latter implies the former). While in most applications involving surfaces parametric values are necessary, it is not obviously the case with curves. It could be the case that most curves used in practice are not very far from an arc-length parametrization, in which case the difference is not important. In curves used for shape design there is no advantage, and some disadvantages, in using a very non-uniform parametrization. Furthermore, if one uses a subdivision process (as we do) the non-uniformity decreases at each subdivision level.

1.3. Speed and Accuracy

In this paper we introduce the use of the Chebyshev basis for subdivision and straightness control, and show how these polynomials provide us with a good recursive adaptive curve drawing method. The Chebyshev polynomials provide an accurate and inexpensive error measure, and the subdivision of curves defined in this basis is quite fast. We will present statistics on over a thousand curves comparing our method to forward and adaptive forward differencing methods, showing our method to be faster for the same accuracy, or more accurate for the same speed.

2. Chebyshev Polynomials

The basis polynomials used with parametric curves are chosen primarily because they facilitate the design process, and additionally because they are easy to evaluate. There is no reason to believe that the same formulation will facilitate the display of the curve. Since the design and the rendering are normally two separate phases, and the rendering is done many times for every design change, there is little or no penalty incurred in a system

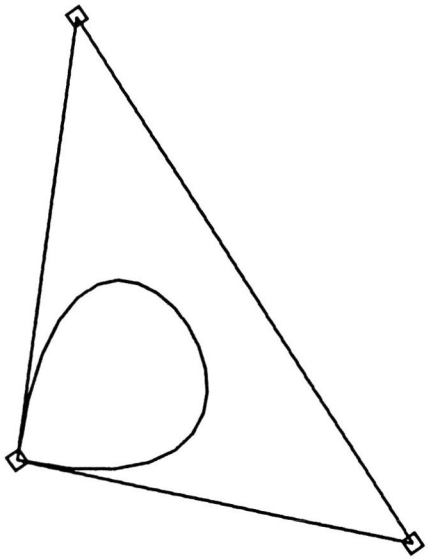

Figure 1.
The wrong first step.

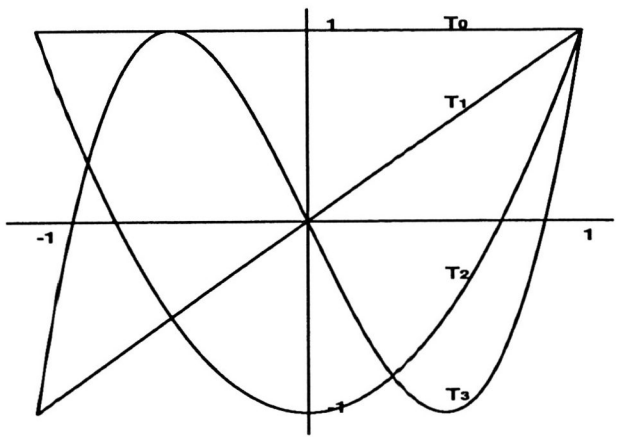

Figure 2.
First 4 Chebyshev polynomials.

if the internal representation uses a polynomial basis different from the one used for design. In fact this is widespread practice: for example systems using β-splines often use Bézier-Bernstein polynomials for rendering, since these give simpler subdivision formulae. It is therefore worth exploring if there is some polynomial basis (we want a basis so that the same space of curves is represented) which would speed up intersection calculations.

2.1. Definition

Chebyshev polynomials are orthogonal polynomials usually denoted $T_n(x)$ such that:

$$T_0(x) = 1$$
$$T_1(x) = x$$

and with the recurrence relation:

$$T_n(x) = 2\ x\ T_{n-1}(x) - T_{n-2}(x)$$

It is immediate that each polynomial $T_n(x)$ is of degree n. A remarkable relation makes apparent many of their interesting properties:

$$T_n(\cos\theta) = \cos(n\ \theta)$$

Figure 2 gives the plot of the first 4 Chebyshev polynomials. The polynomials are best used when the parameter varies in the closed interval [-1,1].

Since Chebyshev polynomials are orthogonal, any polynomial of degree $\leq n$ can be written as a linear combination:

$$P_n(x) = \sum_{k=0}^{n} a_k\ T_k(x)$$

In the case of a polynomial of degree 3, using standard notation:

$$P_3(t)=[t^3\ t^2\ t\ 1\ ][T]\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The 4x4 matrix $[T\ ]$ is:

$$T=\begin{bmatrix} 0 & 0 & 0 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & -3 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

In geometric modeling with parametric curves and surfaces, the parameter(s) range is usually [0,1], so to convert a standard parametric representation from their original basis to the Chebyshev basis, one has to use the following matrix to convert from the [0,1] range to the [-1,1] range:

$$R = \begin{bmatrix} 8 & 0 & 0 & 0 \\ -12 & 4 & 0 & 0 \\ 6 & -4 & 2 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

This matrix premultiplies $[T\ ]$.

To obtain the Chebyshev coefficient as a column vector $[A\ ]$ from any basis representation whose coefficient

column vector is $[P]$ (often called the control points), whose 4x4 matrix is $[M]$ (the unit matrix in the case of the power basis), and whose parameter is in the range [0,1], one has to compute:

$$[A] = [T]^{-1}\,[R]^{-1}\,[M]\,[P] \qquad (1)$$

where

$$R^{-1} = 1/8\begin{bmatrix}1&0&0&0\\3&2&0&0\\3&4&4&0\\1&2&4&8\end{bmatrix} \quad T^{-1} = 1/4\begin{bmatrix}0&2&0&4\\3&0&4&0\\0&2&0&0\\1&0&0&0\end{bmatrix}$$

The product of the 3 matrices has only to be computed once for a given basis. If we call $[C]$ the basis conversion matrix:

$$[C] = [T]^{-1}\,[R]^{-1}\,[M] \qquad (2)$$

then the transformation is:

$$[A] = [C]\,[P] \qquad (3)$$

The basis conversion matrices for the power basis $[C_p]$ and the Bézier basis $[C_b]$ are

$$C_p = 1/32\begin{bmatrix}10&6&6&10\\-15&-6&3&15\\6&-6&-6&6\\-1&3&-3&1\end{bmatrix}$$

$$C_b = 1/32\begin{bmatrix}10&12&16&32\\15&16&16&0\\6&4&0&0\\1&0&0&0\end{bmatrix}$$

2.2. Properties

The basic properties of Chebyshev polynomials are as follows (see[Rivl74] for more details and proofs).

- All the n roots of $T_n(x)$ are real and within the open interval $(-1,1)$. The roots are given by:

$$\xi_{k,n} = \cos\frac{2k-1}{2}n\pi \quad k = 1,\ldots,n$$

- Within the interval $[-1,1]$, $|T_n(x)| \leq 1$. The maxima and minima occur at:

$$x_k = \cos\frac{k}{n}\pi \quad k = 0,\ldots,n$$

where $T_n(x)$ has the values:

$$T_n(x_k) = (-1)^k$$

- The Chebyshev polynomials have the *minimax* property. Again, for any polynomial $P_n(x)$ of degree $\leq n$ we can find a vector $[A]$ of coefficients a_k such that:

$$P_n(x) = \sum_{k=0}^{n} a_k\,T_k(x)$$

If we drop the $T_n(x)$ term from the preceding sum, we obtain a polynomial of degree $\leq n-1$:

$$P^*_{n-1}(x) = \sum_{k=0}^{n-1} a_k\,T_k(x)$$

$P^*_{n-1}(x)$ has the property that of all the polynomials of degree $\leq n-1$ the maximum of the absolute difference:

$$E_n = MAX(|P_n(x) - P^*_{n-1}(x)|)$$

is minimum over the interval $[-1,1]$. Moreover, $E_n = |a_n|$, that is the maximum difference is the absolute value of the coefficient of the Chebyshev polynomial dropped.

The last property is central to the use of these polynomials in boxing and intersection. It means that if we want to replace a polynomial with another of smaller degree, then the way to minimize the maximum error is to convert the polynomial into its Chebyshev representation and drop the Chebyshev polynomial of highest degree.

A problem occurs because in practice we often want to reduce the degree by 2, for example from a cubic to a straight line segment. In general Chebyshev polynomials do not give the best approximation. In the case of a reduction by 2 degrees, the answer are polynomials whose value depends on the ratio of the coefficients of the two highest degree monomials, and are called *Zolotarev* polynomials of order n (if the original polynomial is of degree $n+1$) [Rivl74]. Fortunately Chebyshev polynomials are close approximations of the best, and, for instance, in the case of going from degree 3 down to degree 1,, the Chebyshev approximation is $a_0 + a_1x$, and we have:

$$|P_{best}(x) - (a_0 + a_1x)| \leq |a_2| + |a_3|$$

Since it is so easy to compute the reduced polynomials, and as we will see the cost of not having the best possible polynomial is minor, we fell strongly that Chebyshev polynomial still should be used where they are not theoretically optimal.

3. Subdivision

In many algorithms it will be necessary to subdivide curves. One could of course subdivide in whatever basis the curves are initially represented, or other basis efficient for subdivision, but the conversion to and from the Chebyshev basis would more than offset the savings. Moreover subdividing using the Chebyshev basis directly is relatively easy.

Given a curve C defined by the Chebyshev coefficients (a_0, a_1, a_2, a_3), the following matrices, when applied to the coefficient vector will produce the coefficients of the two half curves:

$$M_{(-1,0)\to(-1,1)} = \begin{bmatrix} 1 & 1/2 & -1/4 & -1/4 \\ 0 & 1/2 & 1 & 3/8 \\ 0 & 0 & 1/4 & 3/4 \\ 0 & 0 & 0 & 1/8 \end{bmatrix}$$

$$M_{(0,1)\to(-1,1)} = \begin{bmatrix} 1 & -1/2 & -1/4 & 1/4 \\ 0 & 1/2 & -1 & 3/8 \\ 0 & 0 & 1/4 & -3/4 \\ 0 & 0 & 0 & 1/8 \end{bmatrix}$$

Since many modern workstations now provide hardware for performing 4 by 4 matrix multiplication this subdivision can be computed using matrices. In the case where this hardware is not available the splitting can be more efficiently computed as follows. If the coefficients of the two sub-curves are called A' and A'', where A' corresponds to the [-1,0] interval in parameter space and A'' corresponds to [0,1], then the computation of the new coefficients is made more efficient by the use of temporary variables:

$$t_0 = a_0/8,\ t_1 = a_3/4,\ t_2 = t_1{*}4,\ t_3 = t_1{+}t_0,$$

$$t_4 = a_2/4,\ t_5 = a_0{-}t_4, t_6 = a_1/2,\ t_7 = t_6{-}t_1,\ t_8 = t_6{+}t_3$$

$$a'_0 = t_0,\ a'_1 = t_4{-}t_1,\ a'_2 = t_8{-}a_2,\ a'_3 = t_5{-}t_7,$$

$$a''_0 = t_0,\ a''_1 = t_4{+}t_1,\ a''_2 = t_8{+}a_2,\ a''_3 = t_5{+}t_7$$

4. Linear Chebyshev approximation.

Given a curve C defined by the Chebyshev polynomials (a_0 , a_1, a_2 , a_3) we define the following quantities:

$$\delta_x = |a_{2_x}| + |a_{3_x}|$$

$$\delta_y = |a_{2_y}| + |a_{3_y}|$$

Since the Chebyshev polynomials T_2 and T_3 are bounded by [-1,1], δ_x and δ_y bound the distance between a point:

$$\mathbf{P}_L = a_0 T_0(t_o) + a_1 T_1(t_o)$$

on the line and its parametric equivalent on the curve:

$$P_C = a_0 T_0(t_o) + a_1 T_1(t_o) + a_2 T_2(t_o) + a_3 T_3(t_o)$$

in the X and the Y direction, respectively. If the Chebyshev linear approximation is replaced with an interpolating line:

$$L = a_0 + a_2 + (a_1 + a_3)t$$

the parametric error or distance between parametric points on the interpolating line and the curve is then bounded by $2\delta_x$ and $2\delta_y$ in X and Y respectively.

The maximum Euclidean distance D between a point on the curve and its parametric equivalent on the line segment is bounded:

$$D \le \sqrt{\delta_x^{\,2} + \delta_y^{\,2}} \le \sqrt{2}\max(\delta_x, \delta_y)$$

5. Stopping criteria

For regular subdivision the requirement is to find a global δ which optimizes both look and speed. In some applications this can be set interactively by the user. If an adaptive subdivision is required then adequate and objective stopping criteria are needed. For adaptive forward differencing the stopping criteria is simply that the next step be no further than a pixel away. We are interested in developing a stopping criteria which correctly identifies sections of the curve which can be approximated by a straight line segment. This criterion is a good one both because most hardware draws straight lines efficiently (but of course *filtered* lines is another story), and because in many application numerous curves have low or zero curvature (in fact are designed to be straight). Since any point on the curve cannot be farther from its corresponding point on the straight line segment than (δ_x , δ_y), it is convenient to define a new error measure on a curve namely $\delta = max(\delta_x, \delta_y)$. The maximum distance D defined above is then bounded by $\sqrt{2}\,\delta$.

The use of the Chebyshev polynomials thus provides us with a measure of deviation from linearity which is close to optimal (and optimal in many cases) and whose computation only costs two additions and a comparison given the Chebyshev coefficients for a span.

6. The algorithm for drawing lines.

The following pseudocode presents the algorithm using adaptive recursive subdivision and a stopping criterion based on Chebyshev basis:

```
Curve = record
  a0, a1, a2, a3
end Curve

DrawCurve(c,ε)
  Curve c; real ε
  if δ(c) < ε then
    /* Draw the interpolating line */
    DrawLine(a0-a1+a2-a3, a0+a1+a2+a3)
  else
    SplitCurve(c,left,right)
    /* SplitCurve applies formulas given in
    ** section 3 to curve c to generate
    ** curves left and right
    */
    DrawCurve(left,ε)
    DrawCurve(right,ε)
  end if
end DrawCurve
```

This method answers both general requirements for line drawing. First it is a fast adaptive method, and second it is capable of displaying curves with little error. It avoids the "self intersecting curve" problem since the error measure is a guaranteed global maximum over the entire span.

Figure 3 shows an example of a curve drawn adaptively with this algorithm, and the same curve drawn by forward differencing at a step corresponding to the smallest step used in the adaptive method.

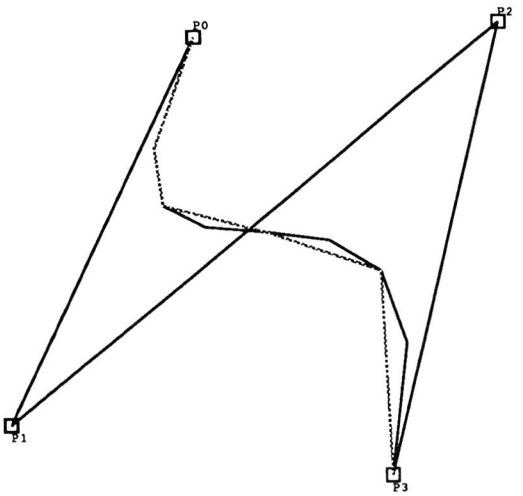

Figure 3.
Cubic curve drawn adaptively and by forward differencing.

7. Statistics

Just claiming a method is better of course does not make it so. Since the relative performance of various approaches depends on the distribution of the characteristics of the curves involved, the results will be somewhat application dependent. A major application using large numbers of cubic curves is font design and display. On display, these curved outlines can be filled or drawn. We will here consider the case where they are drawn. Another paper will address the case where they are filled, since the basic operations are quite different [Buch91]. A single font set provides enough curves to obtain meaningful comparisons. Of course the exact results will vary from fonts to fonts, and if a good predictor of actual performance is desired, the frequency of individual letters and point sizes have to be taken into account.

For our example we used the outline curves for the lower case letters of the Zapf Chancery font (nominally at 12 ps) as provided by the Cubicomp modelling system. The fonts were initially designed by BitStream, and are described in the Cubicomp system as cubic with bias and tension parameters (see [Koch84] for more details). The outline of a few letters is shown in Figure 4.

Figure 4.
Outlines of Zapf Chancery Font.

These were converted to the Chebyshev basis (and to the Bézier basis for later statistics as well) which provided us with 1303 curves (many of them actually straight line segments).

The first set of statistics was calculated by rendering each curve 1000 times (to beat the coarseness of Unix "time" command) using three methods. The first method used was our proposed method. It does not include the time to convert initially to the Chebyshev basis, since in a production system the fonts would be stored in this form. The limit for δ was set at a length equal to one pixel spacing.

The second method used a forward differencing technique with the subdivision level set for all curves by keeping track of the maximum depth necessary in the Chebyshev step. This is necessary to obtain curves that are comparable in quality, since it is the only way for non-adaptive forward differencing to achieve the same precision achieved by the previous method. If *level* is the maximum level of subdivision needed, the Δt for the forward differencing step is then $\Delta t = 1/2^{level}$.

The third method uses adaptive forward differencing to render the curve, with a step always between a half and one pixel. These statistics are presented in table 1

It is clear that the Chebyshev-based method takes on the average 30% of the time taken by forward differencing in our implementation. Adaptive forward differencing is really not in the picture.

To be more comprehensive, two more comparisons are worth making. One is to use the Chebyshev approach (or a similar adaptive scheme) to predetermine the Δt step for forward differencing on a *curve by curve* basis. This is using our approach only as for the linearity criterion. The other interesting comparison is between the levels of subdivision achieved when using our method and using a common method derived from the convex hull property of the Bézier control points. For this purpose we converted the curve representation to Bézier (not charged) and at each level computed the maximum distance from the two "middle" Bézier control points ($\mathbf{P}_1$, $\mathbf{P}_2$) to the line segment $\mathbf{P}_0\,\mathbf{P}_3$. The curve is guaranteed to be found within this limit multiplied by 2/3 (since the maximum weight on the middle points is 2/3). Table 2 gives the results in these two cases, with a repeat of the relevant number for the Chebyshev method.

It is clear that if "assisted" forward differencing can be improved by a factor of about 7 (that is an average of almost three level of subdivision per curve). Of course the cost of finding out what the best level is is not charged to the method in this case, but it could be precomputed and stored which each curve (that would preclude scaling the curves, of course, but in font application this might be realistic).

The comparison with the Bézier test shows that the average level used is about 20% higher, which results in about 20% more work. What makes matter worse for the Bézier solution is that while subdivision is a little faster for Bézier than for Chebyshev (7 shifts and 6 adds for Bézier, 10 shifts and 10 adds for Chebyshev) the computation of δ is quite worse for Bézier (one could have better implementations, but our version uses 20 multiplications, 15 adds/subtracts, 1 square root and 2 to 6 compares for Bézier, versus 2 adds and 5 compares for Chebyshev). In fact the timing shown confirm the large disparity in performance between the two (Bézier on the average takes twice as much as Chebyshev).

8. Conclusions

We have introduced the use of the Chebyshev polynomial basis for fast adaptive recursive drawing of cubic parametric curves. This basis provides us with an accurate and inexpensive error measure for deviation from linearity. We have shown on statistics on a representative sample of more than 1300 curves that it is notably faster than forward differencing in our implementation, and that it gives a better stopping criterion that the use of the Bézier control points.

For a better assessment of a practical implementation, the trade-offs between hardware, firmware, and software computations should be evaluated. As mentioned before, many systems use hardware matrix multiplication for forward differencing. On the other hand the operations of subdivision and computation of δ are easily microcoded in most systems as well. The speed of line drawing vs single pixel writing is also a relevant factor to consider.

The properties of Chebyshev polynomials make them also useful for boxing and intersection, and we are studying these aspects as well. Work currently under way involves the use of the Chebyshev basis to speed up intersections between polynomial parametric curves or surfaces and rays, for filling and trimming in 2D, and ray-tracing in 3D ([Four90 , Buch91]). We are also investigating the use of a similar method to draw filtered curves.

Acknowledgements

We acknowledge the support of NSERC through an operating grant and an equipment grant which considerably facilitated this research. This work was started while both authors were at the University of Toronto, and we acknowledge the support of the Province of Ontario through an Information Technology Research Centre Grant. The support of the University of British Columbia in establishing a computer graphics laboratory in our department and providing grants to equip it and run it is greatly appreciated. We also acknowledge the help of Pierre Poulin, Peter Cahoon and Dave Clement in various aspects of this work.

1024 by 1024				
	Subdivision depth (level)	*Chebyshev method (ms)*	*Forward diff. (ms)*	*Adaptive diff. (ms)*
Average	1.132	0.3043	1.048	8.048
Minimum	0	0.02	1.01	1.03
Maximum	5	2.6	1.22	135.4
Std dev	1.23	0.31	0.01	7.4

Table 1: Timing comparisons for curve drawing.

1024 by 1024					
	Subdivision depth (level)	*Bézier basis (level)*	*Bézier basis (ms)*	*Chebyshev method (ms)*	*Assisted Forward difference (ms)*
Average	1.132	1.378	0.5979	0.3043	0.1358
Minimum	0	0	0.05	0.02	0.04
Maximum	5	5	14.93	2.6	1.04
Std dev	1.23	1.17	0.84	0.31	0.102

Table 2: Comparisons with the Bézier convex hull

References

Buch91 .
J. Buchanan and A. Fournier, "Curve Intersection and Filling Using Chebyshev Polynomials," *In Preparation*, 1991.

Catm75 .
Edwin E. Catmull, "Computer Display of Curved Surfaces," in *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pp. 11-17, Los Angeles, 14 - 16 May 1975.

Chan88 .
M Shantz, S Chang, "Rendering Trimmer NURBS with Adaptive forward Differencing," *Computer Graphics*, vol. 22, 1988.

Fole82 .
James D. Foley and Andries van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, 1982.

Four90 .
A. Fournier and J. Buchanan, "Chebyshev Polynomials for Boxing and Intersections of Parametric Surfaces," *Technical Report, Department of Computer Science, University of British Columbia*, 1990.

Koch84 .
D.H.U. Kochanek and R.H. Bartels, "Interpolating splines with local tension, continuity, and bias control," *Comput. Graphics (USA)*, vol. 18, pp. 33-41, July 1984.

Lane80 .
J.M. Lane, L.C. Carpenter, T. Whitted, and J.F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Comm. of the ACM*, vol. 23, no. 1, pp. 23-34, January 1980.

Prat87 .
Hueue-Ling Lien, M Shantz, V Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces," *Computer Graphics*, vol. 21, 1987.

Rivl74 .
T. J. Rivlin, *The Chebyshev Polynomials*, John Wiley & Sons, 1974.

Rocc89 .
S Chang, M Shantz, R Rocchetti, "Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing," *Computer Graphics*, vol. 23, Dallas, 1989.