

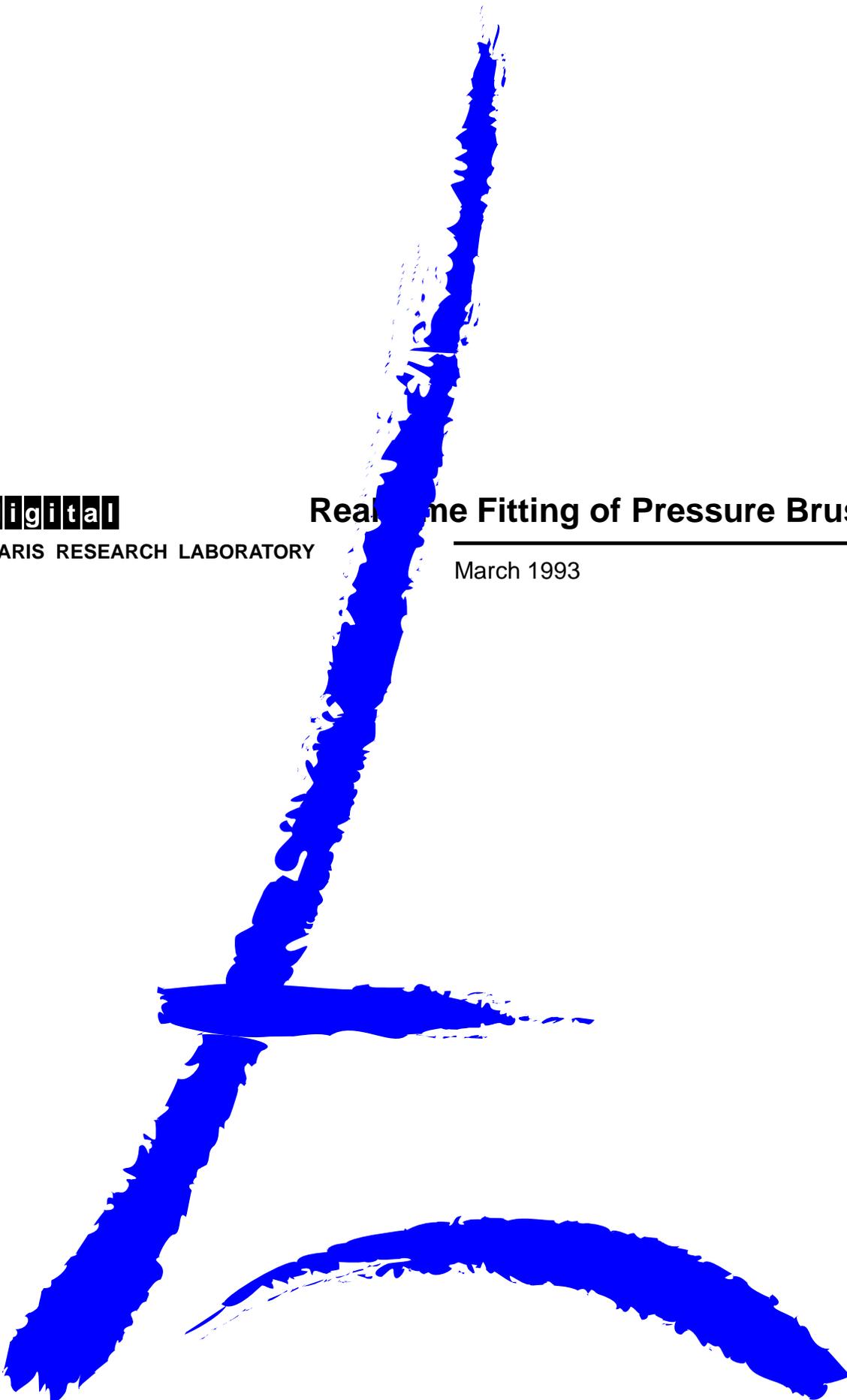
digital

PARIS RESEARCH LABORATORY

Real-time Fitting of Pressure Brushstrokes

March 1993

Thierry Pudet



29

**Real Time Fitting of Pressure
Brushstrokes**

Thierry Pudet

March 1993

Publication Notes

The author can be reached by email at the following address: pudet@prl.dec.com

© Digital Equipment Corporation 1993

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for non-profit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Paris Research Laboratory of Digital Equipment Centre Technique Europe, in Rueil-Malmaison, France; an acknowledgement of the authors and individual contributors to the work; and all applicable portions of the copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the Paris Research Laboratory. All rights reserved.

Abstract

A method is described for fitting the outline of hand-sketched pressure brushstrokes with Bézier curves. It combines the brush-trajectory model, in which a stroke is generated by dragging a brush along a given trajectory, with a fast curve fitting algorithm.

The method has been implemented for a vector-based drawing program in which the user draws with a cordless pressure-sensitive stylus on a digitizing tablet. From the trajectory followed by the stylus, its associated pressure data, and a specified brush, a stroke of variable width is computed and displayed in real time.

First, the digitized trajectory is fitted, thus removing noise. Then, from polygonal approximations of the fitted trajectory and the brush outline, a polygonal approximation of the stroke outline is computed. Working with polygonal approximations reduces computations to simple geometric operations and greatly simplifies the treatment of dynamic, pressure-controlled brushes. Last, the polygonal approximation of the stroke outline is fitted. The result is a closed piecewise Bézier curve approximating the brushstroke outline to within an arbitrary error tolerance.

Several examples of hand-sketched drawings realized with this method are presented.

Résumé

Nous décrivons une méthode permettant d'obtenir le contour d'un trait d'épaisseur variable sous la forme d'une suite de courbes de Bézier. Cette méthode combine le modèle brosse-trajectoire, dans lequel un trait est généré en balayant une brosse le long d'une trajectoire, avec un algorithme de lissage rapide.

Nous avons implanté cette méthode dans un programme de dessin vectoriel. Dans ce programme, l'utilisateur dessine à main levée sur une tablette à digitaliser au moyen d'un stylo sensible à la pression. À partir de la trajectoire du stylo, des pressions qui lui sont associées et d'une brosse de forme donnée, le programme calcule et affiche en temps réel un trait d'épaisseur variable.

Tout d'abord, la trajectoire numérisée est lissée, réduisant ainsi son bruit d'échantillonnage. Dans une seconde étape, une approximation polygonale du contour du trait est calculée à partir des approximations polygonales respectives de la trajectoire lissée et du contour de la brosse. Le fait de travailler à partir des approximations polygonales réduit le calcul à une suite d'opérations géométriques simples, et facilite grandement le traitement des brosses dynamiques dont la taille dépend de la pression. Enfin, l'approximation polygonale du contour du trait est lissée à son tour. Le résultat final est une suite de courbes de Bézier, arbitrairement proche du contour du trait épais.

Plusieurs dessins réalisés suivant cette méthode sont présentés en exemple.

Keywords

Vector graphics, freehand drawing, variable-width stroke, pressure brushstroke, Bézier curves, curve fitting, subdivision, forward differencing, constrained least squares.

Acknowledgements

I wish to thank the courageous artists who tried out our program at an early stage. Credits go to Virginie Jallot and Serge Elissade. I would also like to thank Kathleen Milsted for a careful reading of the manuscript, and Michel Gangnet and Henri Gouraud for their many helpful comments on this paper.

Contents

| | | |
|-----|----------------------------------|----|
| 1 | Introduction | 1 |
| 1.1 | Raster brushstroke | 1 |
| 1.2 | Vector brushstroke | 1 |
| 1.3 | Real time vector brushstrokes | 2 |
| 2 | Least squares curve fitting | 3 |
| 2.1 | Bottleneck | 3 |
| 2.2 | Using higher order Bézier curves | 4 |
| 2.3 | Evaluating the error distance | 5 |
| 2.4 | Curve fitting algorithm | 6 |
| 3 | Brushstroke fitting | 7 |
| 3.1 | Rigid brushes | 7 |
| 3.2 | Getting discrete borders | 8 |
| 3.3 | Fitting discrete borders | 10 |
| 3.4 | Handling dynamic brushes | 12 |
| 4 | Results | 14 |
| 5 | Conclusion | 17 |
| 6 | Appendix | 17 |

1 Introduction

Digitizing tablets with cordless pressure-sensitive styluses are attractive devices for programs that require hand-sketched input, such as paint systems and illustrators. As the user draws on the tablet, the pressure data associated with the trajectory followed by the stylus is used to modify the width of the stroke, therefore simulating a brushstroke.

Existing methods for modeling brushstrokes fall into two classes: those which model some brushstroke attributes at the pixel level and paint the result into a bitmap, and those which model brushstroke outline and rely on scan-conversion for rendering. We refer to these classes as “raster brushstroke” and “vector brushstroke” respectively.

The goal of our method is to achieve, in software, both real time performance and quality graphics results for vector brushstrokes.

1.1 Raster brushstroke

The raster brushstroke approach is based on a digitization process called “brush extrusion”, used mostly in paint programs, where a bitmapped brush is dragged along a trajectory, leaving the image of the brushstroke. Hobby [?] shows how to compensate for the lack of uniform width that happens with straightforward digitization. Whitted [?] describes a technique for anti-aliased strokes using an unchanging textured brush. Paint systems using a pressure-sensitive stylus use the pressure information to dynamically modify some parameters of a circular brush such as its radius or color. Strassmann [?] refines the abstraction of a brushstroke into components whose behavior interact to create the image. Guo [?] and Small [?] model the physical process of ink diffusing into paper fibers in order to achieve realistic rendering effects such as diffuse painting or watercolor.

Raster brushstroke methods are well adapted to real time sketching. This is not surprising as brush extrusion is realized with the help of fast hardwired “bit-blit” operators. Paint systems have taken advantage of this efficiency by incorporating pressure brushstrokes as soon as reliable pressure-sensitive styluses became available. Furthermore, realistic models of paintings have been developed which make brushstrokes more expressive and simulate real paintings. However, raster brushstrokes present two major drawbacks: they are resolution-dependent, and they cannot be edited. Resolution-dependence can be overcome by working at the maximum resolution of all possible output images. In any case, individual strokes cannot be easily, if at all, retouched nor edited.

1.2 Vector brushstroke

In the vector brushstroke approach, the stroke outline is computed from the brush outline and the trajectory. For an elliptical brush and a cubic trajectory, Ghosh and Mudur [?] derive an exact algebraic solution. When the brush is dynamic, no closed form can be obtained. In order to solve the equations analytically for an approximation of the outline, the additional

hypothesis that brush pressure varies slowly compared to trajectory movements must be made. Chua [?] uses cubic Bézier curves to define calligraphic brushstrokes. The control points of each curve of the outline are entered manually. Pham [?] computes the outline as a variable offset approximation of a uniform cubic B-spline trajectory. Each offset knot of the outline must be specified by the user of the system.

Existing vector brushstroke methods present opposite characteristics when compared to raster brushstroke ones. They need heavier computations, which take place in software, and none of them seem well adapted to real time sketching. Solving the equations of the outline analytically is too slow and limited in the range of accessible shapes. Other methods require that the user explicitly enters some mathematical parameters and are thus not suitable for sketching.

On the other hand, all these methods share two significant advantages of vector graphics: resolution-independence and editing capabilities. Strokes can be created, scaled, rotated or flipped very easily. Interactive retouching operations such as recomputing the stroke from the same trajectory using a different brush or different pressure data are straightforward. Moreover, the outline of the stroke is accessible to the user and can be edited like any other shape outline.

1.3 Real time vector brushstrokes

In the context of hand-sketched stroke input for which our method has been designed, real time has the following meaning.

- While dragging the stylus on the tablet, the user sees on the screen a faithful real time echo of the brushstroke, including the variations of width due to the pressure. This can happen at any zoom factor.
- After the stylus is released, the echo is erased, then the brushstroke is fitted and displayed. There must not be noticeable latency due to this computation.

The strategy is to keep the constructive brush-trajectory model but, unlike [?], compute only an approximate analytical representation of the brushstroke outline through least squares curve fitting.

In the next section the bottlenecks of existing least squares curve fitting techniques are identified, and a fast algorithm that uses quintic Bézier curves is proposed. Section 3 describes the application of this algorithm to the construction of brushstrokes obtained with the brush-trajectory model, and also shows how dynamic, pressure-controlled brushes are handled in this context. Finally, Section 4 presents several hand-sketched drawings realized directly on the tablet using our method, and discusses the results.

2 Least squares curve fitting

Least squares curve fitting is a method used for finding an approximate analytical representation of a zero-width digitized trajectory in terms of piecewise parametric polynomials. There are many published methods [?, ?, ?, ?, ?]. Specialized techniques have been employed in design systems for making digital typefaces [?, ?] and illustrations [?, ?].

These methods all have in common the use of cubic polynomials stitched together with some kind of continuity constraints. Except for Plass and Stone [?], who try to produce a nearly minimal number of segments and use a dynamic programming approach not designed for interactive systems, all methods work by trying to fit a single cubic segment to the entire trajectory. After evaluating the error distance, which is the maximum distance between the fitted curve and the digitized trajectory, they stop if this error is less than some specified tolerance, otherwise they divide the trajectory in two at the point of greatest error, and repeat the procedure recursively on both parts until the entire trajectory has been fitted.

2.1 Bottleneck

As shown by Schneider in the cubic case [?], the constrained least squares machinery boils down to solving a 2×2 linear system. However, the bottleneck of the algorithm is more in the repeated evaluation of the error distance needed to decide when to stop the fit.

Given the digitized trajectory \mathbf{s}_i , $i = 0, \dots, I$, and its fitted curve $\mathbf{S}(t)$, $0 \leq t \leq 1$, Plass and Stone compute the error distance D_0 by first, finding the point $\mathbf{S}(\tilde{t}_i)$ of $\mathbf{S}(t)$ that lies closest to sample \mathbf{s}_i , then computing the euclidean distance $\|\mathbf{s}_i - \mathbf{S}(\tilde{t}_i)\|$, and finally taking the maximum over I of those distances, *i.e.* $D_0 = \max_{0 < j < I} \|\mathbf{s}_i - \mathbf{S}(\tilde{t}_i)\|$. For a cubic curve, finding $\mathbf{S}(\tilde{t}_i)$ requires that a fifth degree polynomial equation in t be solved, using, *e.g.*, Newton-Raphson iteration [?].

If D_0 exceeds the specified error tolerance, an iterative technique proposed by Plass and Stone consists in computing a re-parametrization of the samples that will make a single cubic segment fit a greater portion of the trajectory and therefore minimize the overall number of curve segments of the piecewise approximation. To solve the corresponding equations, Newton-Raphson iterations are again needed. The re-parametrization is performed iteratively until no further improvement is detected. A serious drawback of this technique is the need to evaluate the error distance at each step.

The experience of the author is that this algorithm performs well in minimizing the number of curve segments and gives good interactive response time for fitting zero-width curves to hand-sketched trajectories. However, in the context of brushstroke fitting, where not only the digitized trajectory but also the outline is to be fitted, the re-parametrization cycle adds too much of a burden.

2.2 Using higher order Bézier curves

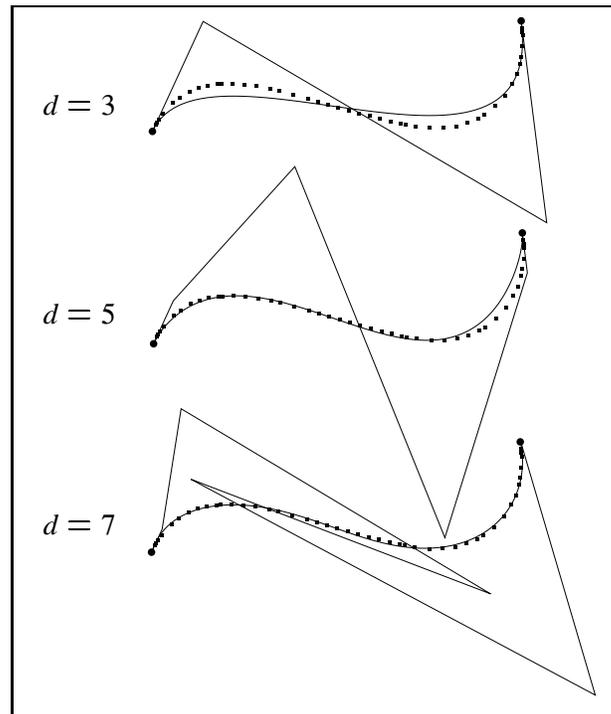


Figure 1: Higher curve degrees provide a comparatively more faithful least squares fit.

The obvious solution that suppresses the re-parametrization has the annoying consequence of producing a piecewise approximation with too many small cubic segments [?]. This somehow defeats the purpose of curve fitting regarding data compaction. To circumvent this problem, our method raises the degree of the fitted curves. Using degree $d > 3$ present the following two characteristics with respect to least squares curve fitting [?].

When the curves are to be fitted with G1 continuity constraints, that is, continuity of the unit tangent vector at each joint [?], the constrained least squares system is a $(2d - 4) \times (2d - 4)$ linear system which can be written in closed form. Solving such a system needs $O(d^3)$ operations. Although in this case the ratio between the cubic case ($d = 3$) and the quintic case ($d = 5$) is 2.7 in favor of the former, it is not a limitative factor compared to the error distance computation.

The re-parametrization cycle is not needed. This is illustrated in Fig. 1 where a single curve was successively fitted without re-parametrization to the same trajectory, keeping the error tolerance constant, and raising the degree from 3 to 7. Only the cases $d = 3$, $d = 5$ and $d = 7$ are shown but the effect is uniform: the higher the degree, the closer the curve to the trajectory. Raising the degree has therefore the same effect as using the re-parametrization cycle [?]. This experimental evidence can be explained by the fact that a quintic Bézier curve

has 2 more control points than a cubic curve, and therefore more freedom to satisfy the same constrained least squares equations. A direct consequence is that although it could still be used to improve the fit as it does for cubics, the re-parametrization cycle can be eliminated when using higher order curves. In practice, there is a balance to be found between raising the degree of curves, the comparatively higher cost of solving the least squares system, and data compaction. Experiments have shown that for doing brushstrokes, quintic curves represent a good tradeoff.

2.3 Evaluating the error distance

Another characteristic of our algorithm is that it uses a different method to evaluate the error distance. The computation is performed incrementally between a polygonal approximation of the curve and the digitized trajectory, and a new termination test is employed.

Let ϵ_{fit} be the error tolerance of the fit. The termination test of [?] is $D_0 \leq \epsilon_{fit}$. We replace it with the test $D_2 \leq \epsilon_{fit}/2$ as follows.

- The error distance $D_0 = \max_{0 < i < I} \|s_i - \mathbf{S}(\bar{t}_i)\|$ is replaced with the maximum distance D_1 between sample s_i of normalized chord-length σ_i , and point $\mathbf{S}(\bar{t}_i)$ having the same arc-length $\sigma(\bar{t}_i)$, $D_1 = \max_{0 < i < I} \|s_i - \mathbf{S}(\bar{t}_i)\|$, where \bar{t}_i is the parameter value such that $\sigma(\bar{t}_i) = \sigma_i$. By definition of D_0 , $D_0 \leq D_1$.

The problem is now to find \bar{t}_i . Formally, $\bar{t}_i = \sigma^{-1}(\sigma_i)$. Instead of solving this equation with numerical methods, we take advantage of the Bézier representation. We compute a polygonal approximation of curve $\mathbf{S}(t)$ to within a certain flatness ϵ_{flat} , using de Casteljau midpoint subdivision [?, ?]. For efficiency, precomputed midpoint subdivision [?, ?] and forward differencing [?, ?] can be used. The subdivision transforms the curve $\mathbf{S}(t)$ into a polygonal approximation \mathbf{S}_k , $k = 0, \dots, K$. We also compute the normalized chord-length σ_k of each point \mathbf{S}_k .

Let \mathbf{S}_{k_i} be the point having the same chord-length σ_i as sample s_i . Since there are only 2 consecutive points \mathbf{S}_{k-1} and \mathbf{S}_k such that $\sigma_{k-1} \leq \sigma_i < \sigma_k$, \mathbf{S}_{k_i} can be linearly interpolated between those 2 points, e.g. $\mathbf{S}_{k_i} = (1 - u)\mathbf{S}_{k-1} + u\mathbf{S}_k$, where $u = \frac{\sigma_k - \sigma_i}{\sigma_k - \sigma_{k-1}}$.

- Let $D_2 = \max_{0 < i < I} \|s_i - \mathbf{S}_{k_i}\|$, and impose $\epsilon_{flat} = \epsilon_{fit}/2$. The termination test $D_0 \leq \epsilon_{fit}$ is replaced with $D_2 \leq \epsilon_{fit}/2$.

By the triangular inequality, $D_1 \leq \max_{0 < i < I} \|s_i - \mathbf{S}_{k_i}\| + \max_{0 < i < I} \|\mathbf{S}_{k_i} - \mathbf{S}(\bar{t}_i)\| \leq D_2 + \epsilon_{flat}$. Thus, imposing $\epsilon_{flat} = \epsilon_{fit}/2$, and $D_2 \leq \epsilon_{fit}/2$ ensures $D_1 \leq \epsilon_{fit}$, hence $D_0 \leq \epsilon_{fit}$.

This method is not limited to quintic curves and can be applied to any parametric polynomial. Its main thrust lies in the evaluation of the new termination test $D_2 \leq \epsilon_{fit}/2$. It can be done incrementally in a very efficient way, by traversing the polygonal approximation and the

digitized trajectory in parallel, evaluating the intermediate distances $\|\mathbf{s}_i - \mathbf{S}_{k_i}\|$ and stopping as soon as the error tolerance is exceeded.

2.4 Curve fitting algorithm

1. Compute the chord-length parametrization for the samples of the digitized trajectory.
2. Estimate the direction of the tangent vectors at endpoints, using a local quadratic interpolant [?].
3. Fit a quintic Bézier curve to the digitized trajectory.
4. Evaluate the goodness of the fit using the incremental technique explained in Section 2.3. If the fit is not faithful enough, divide the digitized trajectory in two pieces and recurse from step 1.

The control flow of the recursive quintic curve fitting algorithm is similar to the one of [?]. Since the tablet provides spatial filtering of the samples, this pre-processing step is not mentioned here¹. Quintic curve segments are then recursively fitted with no re-parametrization, and stitched together with G_1 continuity. Note that since the error distance is evaluated in a lazy way, the digitized trajectory cannot be divided at the point of greatest error. Instead, it is divided at midpoint.

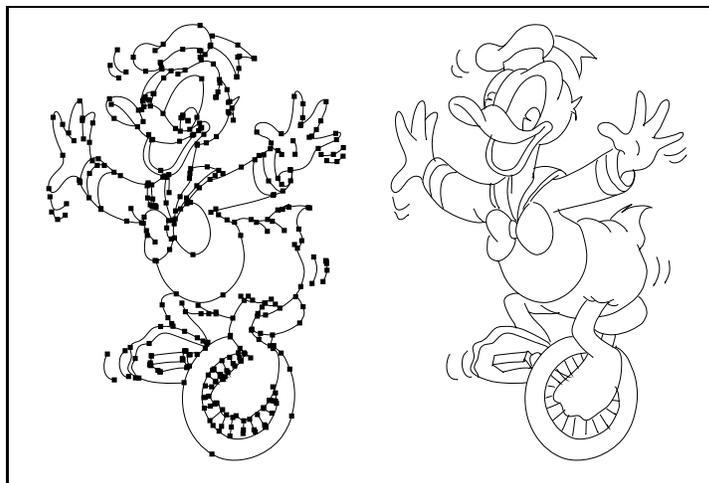


Figure 2: Donald, after Walt Disney: 117 digitized trajectories, 224 quintic curves. Black squares show curve endpoints.

¹Other methods to remove noise such as Gaussian filtering [?] were tried but eventually not used. They modify the initial samples, and even with no further fit, artists using our program found the filtered trajectories not faithful enough.

This algorithm was programmed in C and its performance measured on a DECstation 5000/200 over a data base of cartoon characters drawn by professional artists. The tolerance was set to a constant value of 0.5 mm. Given the high bandwidth of the tablet ($\simeq 200$ pts/sec), depending on the length of the trajectory, and also how rapidly the artist draws, a trajectory can be composed of 20 to 200 samples. For Fig. 2, the mean number of samples per trajectory was 110, and the measured fitting time 20 ms on the average, allowing for real time sketching with zero-width fitted strokes.

3 Brushstroke fitting

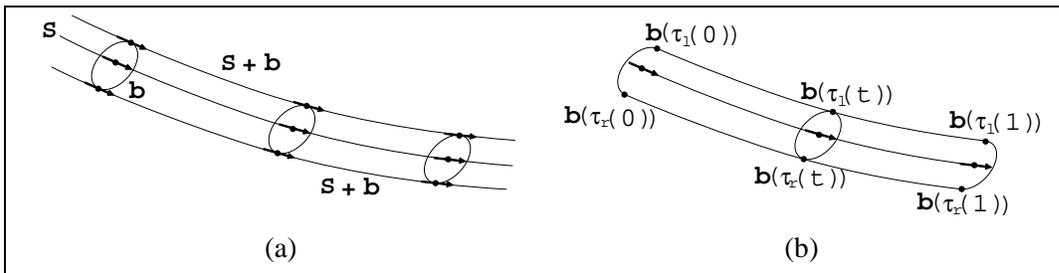


Figure 3: The brush-trajectory model.

In the brush-trajectory model [?, ?], both brush angle and pressure can change dynamically along the trajectory. Dynamic angles reflect more closely the behavior of real brushes used by artists, but is somewhat more difficult to deal with. In this paper, only dynamic pressures are considered.

We distinguish between rigid and dynamic brushes. Rigid brushes ignore pressure and keep a constant size. Dynamic brushes respond to the pressure applied at their tip by changing their size according to some elasticity factor. Rigid brushes are treated first, then dynamic ones. In either case, the brush is assumed to have a convex shape. Non convex brushes are explored in [?], but are computationally too expensive for real time sketching.

3.1 Rigid brushes

A stroke is built by sweeping a convex brush along a central trajectory. The shape of such a stroke is mathematically defined as the envelope of the brush with respect to the trajectory [?]. Formally, if b is the closed outline of the brush and S the central trajectory, both continuous and smooth, the envelope is defined as the sub-set of $S + b$ such that the tangents to the brush b and the trajectory S at those points are parallel, Fig. 3(a).

When the brush and the trajectory are given in parametric form $b(\tau)$ and $S(t)$, $0 \leq \tau \leq 1$, $0 \leq t \leq 1$, this definition translates into a more analytical form. The parameter τ describes the outline of each brush instance while t distinguishes among those instances along the trajectory.

The points of the envelope are solutions of the equation

$$\mathbf{S}'(t) \wedge \mathbf{b}'(\tau) = \mathbf{0} \quad (1)$$

where the operator \wedge denotes the cross product of two vectors, and $'$ the first derivative of a vectorial function with respect to its variable.

For a convex brush translated at position $\mathbf{S}(t)$, there can be only two points $\mathbf{b}(\tau_l(t))$ and $\mathbf{b}(\tau_r(t))$, whose tangents are parallel to $\mathbf{S}'(t)$ and furthermore, those points must be located on each side of the trajectory, Fig. 3(b). By convention, $\mathbf{b}(\tau_l(t))$ is the point to the left of the trajectory and $\mathbf{b}(\tau_r(t))$ the point to its right. The set of points $(\mathbf{b}(\tau_l(t)), 0 \leq t \leq 1)$ defines the left border of the envelope, and symmetrically $(\mathbf{b}(\tau_r(1-t)), 0 \leq t \leq 1)$ its right border. Note that the right border is traversed with decreasing parameter t .

Equation (1) is not solved globally. It is used instead for building the discrete envelope by only solving it *locally* for τ , *i.e.* at each position of the brush along the trajectory. Furthermore, although Equation (1) could be solved locally for τ using numerical methods, it is easier to work directly with the discrete brush.

Given a digitized trajectory and a brush outline, brushstroke fitting works in 3 steps. First, the digitized trajectory is fitted. Then, from polygonal approximations of the fitted trajectory and the brush outline, both obtained through de Casteljau subdivision, a polygonal approximation of the left and right borders of the envelope are computed. Last, those borders are fitted in turn and put together with the starting and ending borders to obtain the brushstroke outline.

3.2 Getting discrete borders

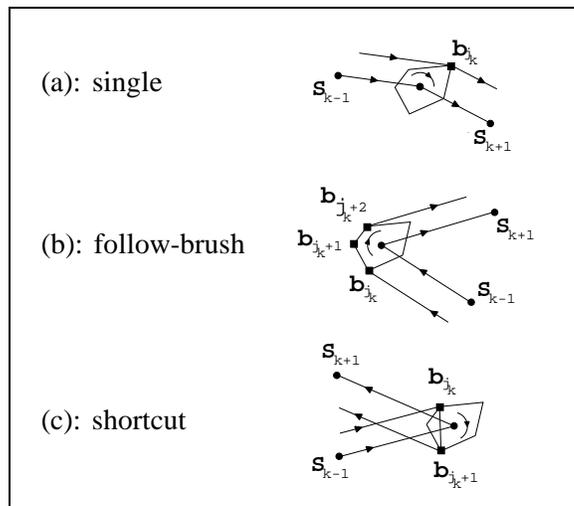


Figure 4: Computing the left discrete border of the envelope. Black squares materialize the discrete envelope.

In what follows, we call

- *discrete trajectory*: the polygonal approximation of the fitted trajectory, obtained through de Casteljau subdivision.
- *discrete brush*: the polygonal approximation of the brush outline, obtained through de Casteljau subdivision.
- *left (resp. right) discrete border*: the polygonal approximation of the left (resp. right) border of the envelope, computed from the discrete trajectory and the discrete brush.

Once the brush and the trajectory are transformed into polygonal approximations, the notion of tangent used in Equation (1) is no longer valid, and must be replaced with the discrete equivalent of *furthest point with respect to a direction*. By an elementary theorem of differential calculus, point $\mathbf{b}(\tau_l(t))$ (resp. $\mathbf{b}(\tau_r(t))$) is also the point of the brush outline lying furthest to the left (resp. right) of the straight line passing through, and tangential to the trajectory at, point $\mathbf{S}(t)$.

The left discrete border is computed from discrete trajectory (\mathbf{S}_k, t_k) , $k = 0, \dots, K$, and discrete brush (\mathbf{b}_j, t_j) , $j = 0, \dots, J$ as follows.

For each discrete position \mathbf{S}_k , $k = 0, \dots, K$,

- The center of the discrete brush is translated at position \mathbf{S}_k .
- Let \mathbf{b}_{j_k} be the point lying furthest to the left of line $(\mathbf{S}_{k-1}, \mathbf{S}_k)$, and $\mathbf{b}_{j_{k+n}}$ the point lying furthest to the left of line $(\mathbf{S}_k, \mathbf{S}_{k+1})$, Fig. 4.
 1. If $n = 0$, $\mathbf{b}_{j_k} = \mathbf{b}_{j_{k+n}}$ (case (a) of Fig. 4), then point \mathbf{b}_{j_k} is added to the discrete border. In this case, there is only one discrete counterpart to point $\mathbf{b}(\tau_l(t_k))$.
 2. Else $n > 0$.
 - (a) The discrete brush being oriented either clockwise or counterclockwise, if the triangle $\mathbf{S}_{k-1}, \mathbf{S}_k, \mathbf{S}_{k+1}$ turns the same way (case (b) of Fig. 4), then the sequence of $n + 1$ points $\mathbf{b}_{j_k}, \mathbf{b}_{j_{k+1}}, \dots, \mathbf{b}_{j_{k+n}}$ is added to the discrete border.
 - (b) Else, the brush and the trajectory locally have opposite orientations, (case (c) of Fig. 4). Only the 2 points \mathbf{b}_{j_k} and $\mathbf{b}_{j_{k+n}}$ are added to the discrete border. This means that instead of following the brush, a shortcut is taken between those 2 points. The convexity of the brush guarantees that the shortcut will not cross the brush boundary.
In addition, both points are marked as breakpoints. These marks will be interpreted when the discrete border is fitted.

The right discrete border is obtained in a similar way, but traversing the discrete trajectory downward from \mathbf{S}_K to \mathbf{S}_0 .

Thanks to brush convexity, the previous algorithm can be implemented in a very efficient way. For any point on a convex outline, the furthest distance with respect to a given direction is

a function that increases from 0 to a maximum value, then decreases again towards 0. The implementation takes advantage of this property by starting the search for the next point of the discrete border with the point of the discrete brush that was found at the previous step, going next to its neighbor, and evaluating the distance until a local maximum is reached. Instead of being proportional to $K \times J$, as the above description would suggest, it makes the overall process almost linear in the number K of points in the discrete trajectory.

3.3 Fitting discrete borders

Applying the curve fitting algorithm of Section 2.4 to the left and right discrete borders in sequence is straightforward. The only difference is that now, the breakpoints set at sampling time to mark shortcuts are interpreted as discontinuities. Such points prevent the entire border from being fitted at once. Instead, intermediate portions between two breakpoints are fitted independently, with only positional continuity (C_0) at the joints.

The situation at trajectory endpoints is somewhat different. The starting and ending borders of the envelope need not be fitted since they are just portions of the brush outline. The starting border is the portion of the brush outline going from the first point $\mathbf{b}(\tau_l(0))$ of the left border to the first point $\mathbf{b}(\tau_r(0))$ of the right border, Fig. 3(b). Similarly, the ending border is the portion of brush outline going from $\mathbf{b}(\tau_l(1))$ to $\mathbf{b}(\tau_r(1))$. If the brush outline is given in Bézier form, starting and ending borders are easily obtained through de Casteljau subdivision.

By definition, the fitted left border goes from $\mathbf{b}(\tau_l(0))$ to $\mathbf{b}(\tau_l(1))$ and the fitted right border from $\mathbf{b}(\tau_r(1))$ to $\mathbf{b}(\tau_r(0))$. Connecting starting border to left border to ending border to right border in this order defines an approximate analytical representation of the brushstroke outline to within a specified error tolerance.

The need to go from digitized trajectory to fitted trajectory then immediately back to discrete trajectory, and to compute the discrete borders thereof, may seem unclear. A more direct solution would be to get the discrete borders from the digitized trajectory. This was tried, but because of the noise associated with the raw data, it did not give good results, especially when small brushes were used. The digitized trajectory had therefore to be fitted first. If more than spatial filtering is done at pre-processing time, this step may be omitted.

Fig. 5 shows the result of the algorithm for a rigid circular brush. Of course, the error tolerance can always be set manually, but in most cases, a default value equal to some fraction of the brush diameter (say 1/30) gives visually good results. In part (a) of Fig. 5, both the central trajectory and the outline are stroked. In part (b), the outline is stroked and the curve endpoints are shown. There are two breakpoints located on the right border of the envelope where the central trajectory bends sharply. In part (c), the outline is filled, using the non-zero winding rule. Note that the non-zero winding rule is mandatory for filling brushstrokes. The even-odd rule would not do for strokes with self intersections. The reason is that, by construction, all the points covered by the brush during the sweep, and defining the interior of the envelope, have a non zero winding number.

Other examples presented in Fig. 6 are brushstrokes built with an elliptical brush held at

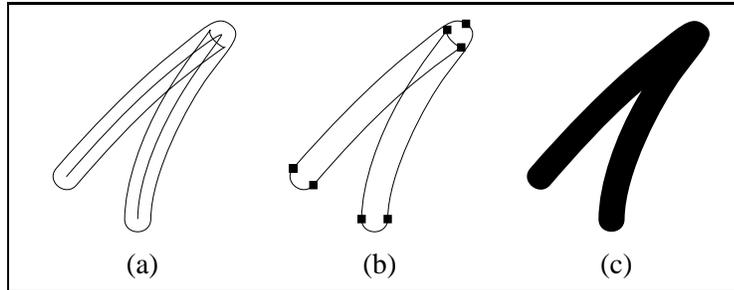


Figure 5: Brushstroke from a rigid circular brush.

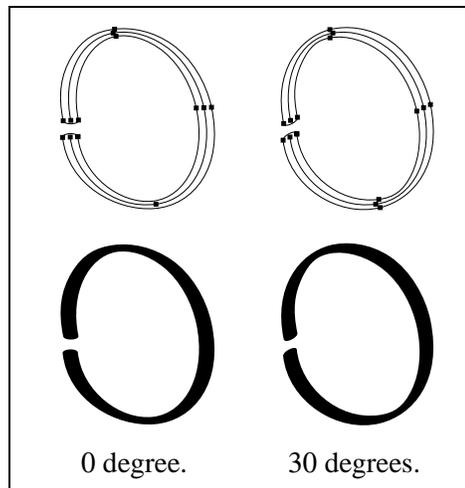


Figure 6: Brushstrokes from the same elliptical rigid brush held at different angles.

different angles. Even with no pressure, a non circular brush always produces a stroke of varying width. Here, the strokes were computed from the same trajectory.

3.4 Handling dynamic brushes

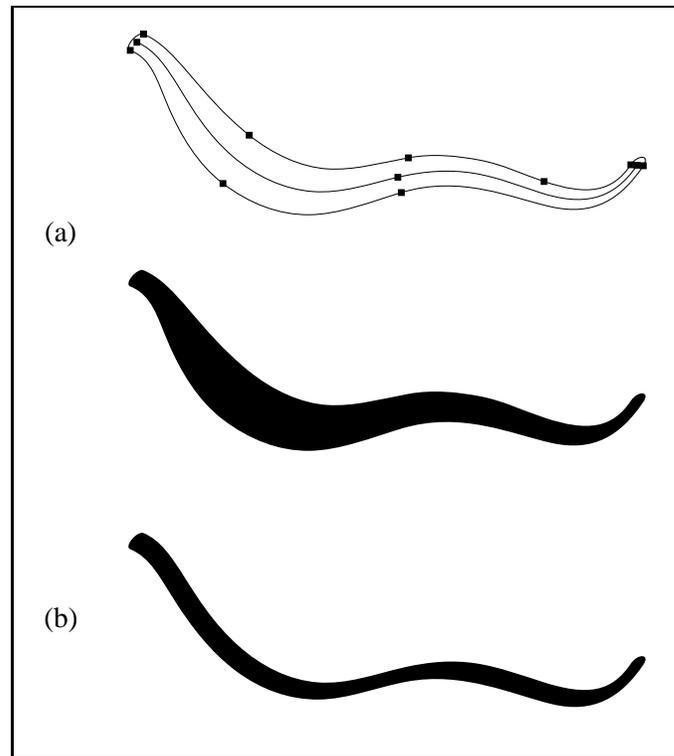


Figure 7: Brushstroke from a dynamic elliptical brush (a), and, for comparison, the stroke obtained with the same but rigid brush (b).

A dynamic brush has an associated elasticity e . If no pressure is applied, the brush keeps its natural size. For simplicity, pressure values are normalized between 0 (no pressure) and 1 (highest pressure). Given some pressure p , the brush dilates according to e and p . The elasticity gives the maximum allowed dilatation with respect to the natural size. For instance, a dynamic circular brush of initial diameter ρ can dilate to a circle of diameter $e \rho$.

The brush always starts and finishes its travel along the trajectory with pressure 0 since at those positions, the pressure-sensitive stylus has to be released. The corresponding brushstroke therefore begins at position $\mathbf{S}(t = 0)$ and finishes at $\mathbf{S}(t = 1)$ of the trajectory with the natural size of the brush.

At intermediate positions $\mathbf{S}(t)$, pressure values $p(t)$ are converted into scaling factors $s(t)$. Each scaling transformation is centered at the center of the brush, and applies to the brush outline. It is required that $s(0) = s(1) = 1$ (no scaling) and also that when the highest

pressure is applied, $s = e$. The simplest model is to interpolate linearly between $p(0)$ and $p(1)$, i.e. $s(t) = 1 - (1 - e)p(t)$, but non linear variations are also possible. This definition implies that a dynamic brush with elasticity $e = 1$ is, in fact, a rigid one.

However, the pressure value $p(t)$ at position $\mathbf{S}(t)$ of the fitted trajectory is not known *a priori*. It is true that the stylus associates with each sample \mathbf{s}_i of the *digitized trajectory* a pressure value p_i , but since the discrete borders are computed from the *fitted trajectory*, this correspondence cannot be used directly.

The problem is then to map pressures p_i to the fitted trajectory in a coherent way. To do this, we associate to pressure p_i the normalized chord-length σ_i of sample \mathbf{s}_i . This gives us a discrete pressure profile (p_i, σ_i) that we map to the fitted trajectory $\mathbf{S}(t)$ through arc-length. In fact, we have already solved the problem in Section 2.3 when computing the error distance. The only difference is that here, pressure values p_i are substituted for samples \mathbf{s}_i .

Given a digitized trajectory $\mathbf{s}_i, i = 0, \dots, I$, associated pressure value p_i , and corresponding fitted trajectory $\mathbf{S}(t), 0 \leq t \leq 1$.

1. Compute normalized chord-lengths σ_i of samples $\mathbf{s}_i, i = 0, \dots, I$.
2. Compute the discrete trajectory $(\mathbf{S}_k, t_k), k = 0, \dots, K$ of $\mathbf{S}(t)$ through de Casteljau subdivision, and compute also the normalized chord-lengths σ_k of points $\mathbf{S}_k, k = 0, \dots, K$.
3. For each point \mathbf{S}_k , compute value p_{i_k} having the same normalized chord-length σ_k as \mathbf{S}_k :

$$p_{i_k} = (1 - u)p_{i-1} + u p_i,$$

$$u = \frac{\sigma_i - \sigma_k}{\sigma_i - \sigma_{i-1}},$$

where p_i is the unique value of the pressure profile such that $\sigma_{i-1} \leq \sigma_k < \sigma_i$.

Figure 8: Mapping pressure profile to fitted trajectory.

This method still works even if the pressure data do not come from the stylus. In fact, any polynomial function $f(t)$ can be mapped to the trajectory $\mathbf{S}(t)$. All that is needed in this case is an additional step of de Casteljau subdivision to get the corresponding polygonal approximation.

An example of a pressure-controlled brushstroke is shown in Fig. 7. A dynamic elliptical brush of elasticity $e = 4$ was used.

4 Results

We have implemented the method in a prototype vector-based drawing program in which the user draws with a cordless pressure-sensitive stylus on the tablet. The following brush parameters can be set: width, height, angle, and elasticity. Brush sizes vary between 1/10 mm and 10 mm. There are two separate controls for setting tolerances: one is for the error tolerance tol_t for fitting the digitized trajectory, the other error tolerance (flatness) tol_b for approximating the brushstroke outline. The latter is typically smaller.

While drawing, the user gets a real time echo of the stroke displayed on the screen. For each new sample s_i , a polygonal approximation of the envelope of the brush with respect to the straight line segment (s_i, s_{i-1}) is computed, using the technique described in Section 3. The echo is displayed by painting the resulting overlapping polygons in sequence. This gives a result equivalent to brush extrusion. No fitting is done at this stage. After the stylus is released, the previous echo is erased, then the brushstroke is fitted and displayed.

Ultimately, the performance of curve fitting methods is tied to the number of samples of the digitized trajectory. Not surprisingly, our algorithm can show poor performance for exceptionally large trajectories (say > 800 samples). In practice, this is not often the case, with the notable exception of the “roughs” drawn by cartoon cell animators before cleaning. In such cases, the trick is to cut the digitized trajectory into several pieces and to fit them successively. Because of the error distance computation, this is faster than fitting the whole trajectory at once. Furthermore, the brush-trajectory model ensures that the same outline is obtained in both cases. Although this trick was not implemented, the artists who realized the drawings presented here could work on the tablet at their paper speed, without having to wait or loosing data.

We have tried to exercise the algorithm and measure its performance on different styles of drawings, including comic strips and cartoon cell design as well as cursive calligraphy. All timings refer to a DECstation 5000/200. Each drawing is also shown full page in Appendix 6. Outputs are done on a black-and-white laser printer at 300 dots per inch.

Elliptical brushes are most often used for calligraphy. In Fig. 9, the author used an elliptical rigid brush of size $90/10 \times 20/10$ mm, setting $tol_t = 3/10$ mm and $tol_b = 1/10$ mm. The character, whose real height is about 25 cm, is built from a single digitized trajectory of 231 samples, fitted with 6 quintics in 50 ms. The outline is composed of 29 curves fitted in 300 ms. With true calligraphers at work, such input may be used in font design software as a first sketch to be tuned later on.

Elliptical brushes can also provide pleasing results for drawings. Fig. 10 was realized with a very flat elliptical brush of size $12/10 \times 1/10$ mm, oriented at 60 degrees, and with tolerances $tol_t = 5/10$ mm and $tol_b = 1/10$ mm. There are 88 digitized trajectories, and 843 curves accounting for the brushstrokes outlines. The total time for fitting all the strokes is 7 s. On the average, each brushstroke is fitted in less that 80 ms, allowing for real time hand-sketching.

Fig. 11 presents an example drawn with a dynamic circular brush. Unlike the previous

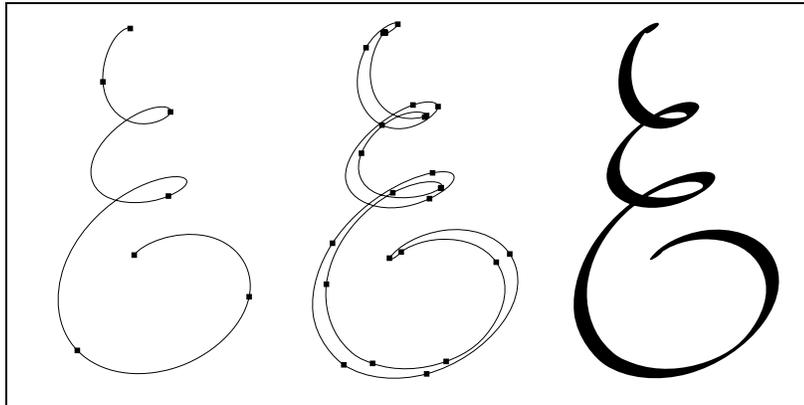


Figure 9: Hand-drawn calligraphic E.



Figure 10: Boxing lobster.



Figure 11: Iznogoud from Tabary.



Figure 12: Sans titre.

examples, the artist changed the size of the brush and its elasticity in the middle of the drawing. This is visible for the fat strokes of the nose. The moustache and the beard were done by scribbling with the stylus. There are 265 digitized trajectories, and each brushstroke is fitted in about 95 ms on the average.

For Fig. 12, the artist used different circular brushes with different colors, which makes the drawing look more like a (digital) painting rather than typical line art. [put serge.ps back]

5 Conclusion

We have presented a method for fitting pressure brushstrokes. Our goal is to get real time performance and quality visual results for graphics arts applications.

In order to define an approximate analytical representation of the brushstroke outline to within an arbitrarily small error tolerance, we compute a polygonal approximation of the brushstroke outline, to be fitted with quintic Bézier curves. Working with polygonal approximations greatly simplifies the treatment of dynamic, pressure-controlled brushes. We have developed an incremental technique for evaluating the error distance in the termination test of the curve fitting algorithm. Quintic curves are chosen as a compromise between time and space, but cubic curves can also be used. This gives us a uniform, resolution-independent model of a brushstroke for building hand-sketched vector drawings.

The method has been implemented in a program which was used by professional artists. Several drawings are presented.

So far, we have not investigated the rendering aspect of vector brushstrokes. It is conceivable that the techniques developed for realistic rendering of raster strokes could be adapted and used through procedural rendering to give vector brushstrokes a comparable artistic expressiveness. We are also interested in taking advantage of the the vector-based representation of the stroke in order to achieve other effects.

6 Appendix

This section presents the drawings of Section 4 as well as some others, in fullpage output.

PRL Research Reports

The following documents may be ordered by regular mail from:

Librarian – Research Reports
Digital Equipment Corporation
Paris Research Laboratory
85, avenue Victor Hugo
92563 Rueil-Malmaison Cedex
France.

It is also possible to obtain them by electronic mail. For more information, send a message whose subject line is `help to doc-server@prl.dec.com` or, from within Digital, to `decprl : : doc-server`.

Research Report 1: *Incremental Computation of Planar Maps*. Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. May 1989.

Research Report 2: *BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic*. Bernard Serpette, Jean Vuillemin, and Jean-Claude Hervé. May 1989.

Research Report 3: *Introduction to Programmable Active Memories*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. June 1989.

Research Report 4: *Compiling Pattern Matching by Term Decomposition*. Laurence Puel and Ascánder Suárez. January 1990.

Research Report 5: *The WAM: A (Real) Tutorial*. Hassan Aït-Kaci. January 1990.

Research Report 6: *Binary Periodic Synchronizing Sequences*. Marcin Skubiszewski. May 1991.

Research Report 7: *The Siphon: Managing Distant Replicated Repositories*. Francis J. Prusker and Edward P. Wobber. May 1991.

Research Report 8: *Constructive Logics. Part I: A Tutorial on Proof Systems and Typed λ -Calculi*. Jean Gallier. May 1991.

Research Report 9: *Constructive Logics. Part II: Linear Logic and Proof Nets*. Jean Gallier. May 1991.

Research Report 10: *Pattern Matching in Order-Sorted Languages*. Delia Kesner. May 1991.

Research Report 11: *Towards a Meaning of LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, October 1992).

Research Report 12: *Residuation and Guarded Rules for Constraint Logic Programming*. Gert Smolka. June 1991.

Research Report 13: *Functions as Passive Constraints in LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, November 1992).

Research Report 14: *Automatic Motion Planning for Complex Articulated Bodies*. Jérôme Barraquand. June 1991.

Research Report 15: *A Hardware Implementation of Pure Esterel*. Gérard Berry. July 1991.

Research Report 16: *Contribution à la Résolution Numérique des Équations de Laplace et de la Chaleur*. Jean Vuillemin. February 1992.

Research Report 17: *Inferring Graphical Constraints with Rockit*. Solange Karsenty, James A. Landay, and Chris Weikart. March 1992.

Research Report 18: *Abstract Interpretation by Dynamic Partitioning*. François Bourdoncle. March 1992.

Research Report 19: *Measuring System Performance with Reprogrammable Hardware*. Mark Shand. August 1992.

Research Report 20: *A Feature Constraint System for Logic Programming with Entailment*. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. November 1992.

Research Report 21: *The Genericity Theorem and the notion of Parametricity in the Polymorphic λ -calculus*. Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. December 1992.

Research Report 22: *Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite*. François Bourdoncle. January 1993.

Research Report 23: *Dessin à main levée et courbes de Bézier : comparaison des algorithmes de subdivision, modélisation des épaisseurs variables*. Thierry Pudet. January 1993.

Research Report 24: *Programmable Active Memories: a Performance Assessment*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. March 1993.

Research Report 25: *On Circuits and Numbers*. Jean Vuillemin. April 1993.

Research Report 26: *Numerical Valuation of High Dimensional Multivariate European Securities*. Jérôme Barraquand. March 1993.

Research Report 27: *A Database Interface for Complex Objects*. Marcel Holsheimer, Rolf A. de By, and Hassan Aït-Kaci. March 1993.

Research Report 28: *Feature Automata and Recognizable Sets of Feature Trees*. Joachim Niehren and Andreas Podelski. March 1993.

Research Report 29: *Real Time Fitting of Pressure Brushstrokes*. Thierry Pudet. March 1993.

digital

PARIS RESEARCH LABORATORY
85, Avenue Victor Hugo
92563 RUEIL MALMAISON CEDEX
FRANCE