A Mark-Based Interaction Paradigm for Free-Hand Drawing

Thomas Baudel
LRI, Université de Paris-Sud, CNRS URA 410
91405 Orsay Cedex, France
e-mail: thomas@lri.fr
http://www-ihm.lri.fr/~thomas/

ABSTRACT

We propose an interaction technique for editing splines that is aimed at professional graphic designers. These users do not take full advantage of existing spline editing software because their mental representations of drawings do not match the underlying conceptual model of the software. Although editing splines by specifying control points and tangents may be appropriate for engineers, graphic designers think more in terms of strokes, shapes, and gestures appropriate for editing drawings. Our interaction technique matches the latter model: curves can be edited by means of marks, similar to the way strokes are naturally overloaded when drawing on paper. We describe this interaction technique and the algorithms used for its implementation.

KEYWORDS: Mark-based interaction, Gestures, Spline editing, Interaction models, Graphic design, CAD.

INTRODUCTION

Graphic designers and other art professionals are increasingly likely to use professional drawing software based on spline editing, such as Adobe Illustrator®. Compared to conventional paper-based design tools, these applications make it easier to reuse graphic components and layouts, enable easy and clean corrections and encourage the draft-based and iterative design methods already widely used with paper. Intensive training and good mastering of hand-drawing techniques is not required to use these applications, so non-skilled drawers can produce decent-quality graphics.

However, many (most ?) skilled designers are still reluctant to use such software exclusively. Even computer-literate graphic designers find it easier to sketch first on paper and then digitize their final drafts rather than use the editing facilities provided by computerized design tools. This is so common that some specialized software for cartoonists doesn't provide any partial editing functions at all [9]. This design decision is based on careful observation of the target users:

• Most cartoonists are trained to draw the right shape on the first try. They learn to avoid making mistakes because they are too costly to repair on paper.
• When dissatisfied with a particular curve, professional cartoonists usually prefer to redraw it. This preserves a clean drawing style and the spontaneity of their strokes.
• Providing editing facilities requires the introduction of modes and burdens the application with rarely-used functions.

Current methods for editing spline curves consist of moving control points and tangents to the curve. These editing methods were invented and targeted for engineers and industrial designers, in a specific context [5]:

• The representation of curves matches the constraints of the objects to be represented with CAD packages: it provides perfect control over the positions of the control points of the curves, and provides useful geometric properties, e.g., C^2 continuity where needed and easy computation of bounding surface or volume.
• The target users usually have some background in mathematics enabling them to master easily the peculiarities of the interaction technique. For example, when a control point is near an anchor point, small moves of one or two pixels produce very important changes in the overall shape of the curve, because the tangent to the curve at the anchor point is affected by every discrete increment.

Although this spline-editing technique may be appropriate for CAD engineers, it has proven difficult for graphic designers to use because it does not fit the conceptual model they have of their tasks.

We propose a mark-based interaction technique that is specifically intended for graphic designers. Our design is based on patterns extracted from their existing drawing and editing techniques rather than on a data representation focused on mathematical models and computer management of geometric data. Our interaction method allows users to edit splines with marks and pen input, based on common drawing and editing techniques on paper. Although easier for graphic designers to understand, this approach requires translating subjective aspects of marks into a geometric data format that can be manipulated by the computer but is still tractable with current hardware and software.

After placing our work into context, we present our observations and the concepts that led to this interaction method. We then describe the algorithms we used and possible enhancements. Finally, we propose a preliminary evaluation of the benefits of this interaction technique. In conclusion, we suggest further enhancements to this interaction paradigm that will improve the tools graphic designers or artists use to create and edit drawings.

RELATED WORK

Many interaction techniques and spline representations providing more "natural" ways to edit curves have been proposed. Most work has focused on new representations of curves: different definitions of the relationship between the curve and its control-points, e.g., B-Splines [14]) or the addition of parameters such as "tension" or "bias", e.g., ß-splines [2]. Various formulations have been evaluated. For example, Bartels et al. [3] use a shape-matching task to evaluate the respective qualities of five different spline formulations: B-Spline, Bézier, Catmull-Rom, "end-condition" and "natural" curves.

More recently, Fowler and Bartels [10] proposed a technique for direct manipulation of splines that permits moving *any* point on the curve instead of specific control-points. This technique requires the introduction of constraints to specify invariance of positions or tangents to the curves, but should lead to more natural interaction techniques. Some products such as Adobe Illustrator® or Canvas® already provide a limited version of this functionality, in addition to conventional techniques. However, mixing editing techniques within a single product may end up confusing rather than helping the end user.

Each of these representations assumes the use of a mouse as an input device for creating and moving control-points. Professional tools for graphic designers often use pen input on a graphics tablet. Curves are drawn directly and the sampled trajectory is transformed into an adequate representation using a curve-fitting algorithm [1], [16]. For graphic designers, who have mastered drawing by hand and don't focus on the geometric properties of their drawings, this technique is probably the best way to create a new curve because of its direct style. However, to our knowledge, no correspondingly direct interaction technique exists for editing and transforming an already-drawn curve.

Many mark-based interaction techniques have been developed to allow for direct input of commands with a pen, such as the PenPoint [8] or the Apple Newton user interfaces. General algorithms [15] for these techniques are widespread. Usually they are shortcuts for structured commands, rather than complex editing of objects. The shape of each mark is more or less directly linked to a single command and variations in size, orientation or location are sometimes used as simple parameters ([11], [12], [4]). We use marks as a concise way to enter complex reparametrization data rather than as command shortcuts.

A MARK-BASED INTERACTION TECHNIQUE

We believe that the difficulties faced by graphic designers with conventional spline editing software is based on a mismatch between the software's conceptual model and the representation of a drawing by a skilled graphic designer. Instead of thinking of a curve as a list of control points, a designer thinks in terms of strokes, global balance of shapes, gestures to be made or graphic techniques that give a different rendering. It is not easy to provide a formal model and computer representation that deals with these kinds of concepts in a general way. Rather than searching for such a model, we observed and exploited existing techniques to solve the problem of editing an already drawn curve.

An example from the real world

The following sketch of a bottle (figure 1), was made by a poor sketcher. Since the first stroke did not show a balanced bottle, the designer redrew or "overloaded" some strokes on places where the original did not look satisfying. The result is a better balanced general impression of a bottle, although the drawing is not clean.

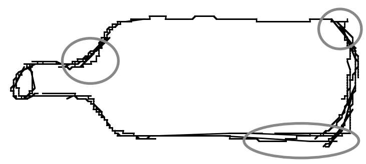

Figure 1: Strokes overloading the bottle's initial contour (in gray ellipses)

Skilled designers are trained, with some difficulty, to avoid using this technique, because of the resulting heaviness of the drawing. However, it is a very natural interaction technique for editing curves: once in "edit" mode, strokes can be interpreted as shape modifiers instead of new curves. Figure 2 shows the base of the letter "i" in Times Roman typeface: successive strokes "pull" the base of the letter to make a larger serif.

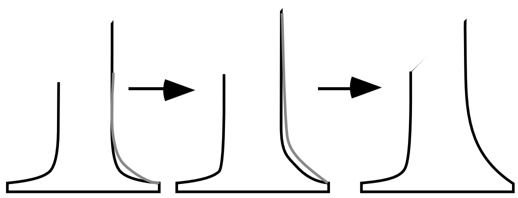

Figure 2: Editing a curve with successive correcting strokes (in gray)

We use stroke recognition techniques, not as a way to issue commands, but rather to provide complex modification and reparametrization patterns in a way that could not be specified easily with a simpler input scheme.

This technique relies upon and enforces a natural means of editing curves that should be easy to learn and use. Furthermore, it allows for global as well as local reparametrization in a single stroke. A designer can start with large creation and reparametrization strokes, to obtain a global outline of his or her work quickly, and then make local changes to refine the drawing. This top-down approach is not available with conventional editing techniques, where the designer must specify the control-points one by one, not too far from their exact location, and cannot test radical changes with a few simple actions.

Sample application

We built a testbed for this interaction technique that provides an efficient interaction method for creating simple curves and 2D shapes made of paths. Our application is implemented on an HP9000/755 workstation and uses a Wacom UD1212 graphics tablet, which allows fast (205 points/second) and precise (±0.15 mm) tracking of gestures, as well as multi-threaded entry with a cursor and a stylus. Using the keyboard, the user selects among four modes to edit a drawing (figure 3): creation of a curve (free-hand drawing), modification or suppression of a portion of curve, and joining of two curves (or closing a path).

Sometimes, it may not be fully clear which curve is the target of the modifying stroke (figure 4). In such cases, a sub-mode is used to alleviate possible ambiguities: when the system detects several possible targets, it highlights the portions of curves that make good candidates. The user can then point at the desired target for completion of the command.

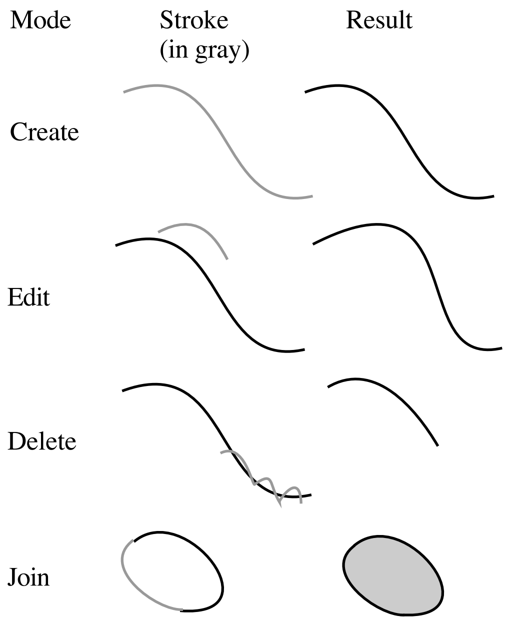

Figure 3: The effect of a stroke in each of the four modes. (Join mode is currently not fully implemented)

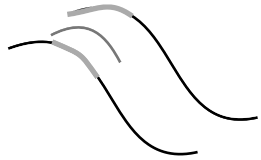

Figure 4: Here, two curves can be reparametrized. Both candidate portions are highlighted, the user can pick which one will be modified.

Implementing a whole application from our sample can be done either by using conventional interaction widgets such as menus and palettes, or more advanced techniques such as see-through tools [6], [7]. We have considered using gesture recognition techniques such as Rubine's algorithm [15] to differentiate between "Delete" and "Edit" strokes automatically, or use contextual information to unify "Join" and "Edit" modes. But such techniques may cause confusion and may prevent users from freely drawing any shape without their gestures being misinterpreted. We have chosen to clearly identify modes, e.g., by changing the shape of the pointer, to reflect the exact semantics of the strokes that are to be drawn and avoid conflicts between the users' expectations and the system's interpretation.

IMPLEMENTATION

In this section, we detail the implementation of the reparametrization algorithms ("edit" mode). The "create" mode is implemented using Schneider's algorithm [16]. The "delete" and "join" modes can be derived easily from the following algorithms. We do not provide a detailed description of how to handle closed paths, since they do not require a reformulation of the principles but only careful implementation.

Notation

Our underlying formalization requires curves to be represented by arc-length parametrized functions. A drawing is made of a set F of parametric curves. Each curve f of F is made of 2 real functions of a parameter varying in $[0..L_f]$, L_f being the length of the curve: $\forall$ t $\in [0..L_f]$, $f(t) = (f_x(t), f_y(t))$ where f_x and f_y can be any continuous functions, such as sequences of Bézier patches or polylines. They need not necessarily be differentiable everywhere, to allow curves with sharp corners to be handled like other curves. f_{ab} denotes the restriction of f to the interval $[a..b] \subseteq [0..L_f]$ (or $[b..a]$ if $b \geq a$). A stroke m, using the same representation: $\forall$ t $\in [0..L_m]$, $m(t) = (m_x(t), m_y(t))$, is drawn as a modifying mark.

We propose a general algorithm that transforms curve f into resulting curve r :

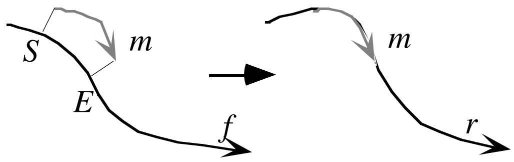

Figure 5: Visual aspect of the reparametrization of a curve.

We decompose the operation into two steps:

• Find the target f in F and the parameters S (for *start of match*) and E (for *end of match*) in $[0..L_f]$ that provide the best match with m. This implies defining a distance between portions of the curves that matches the users visual perception of them. Note that when m and f are drawn in opposite directions, we have $E \leq S$.

• Transform f into r, given S, E, the characteristics of m and the current mode. Several methods can be used, with no definitive advantage yet found to favor one.

Finding the target curve

Finding a target for reparametrization requires finding the portion of a curve that is "visually the closest" to m. "Visually closest" is a subjective term that can hardly be defined formally. Our mathematical definition has been chosen after several trials, and matches convincingly our objective for each of the particular cases we have experimented with. It must be noted that special cases such as the one described in figure 6 will tend to be avoided naturally, not because of the inability of our algorithms to handle them (actually, they will interpret it), but because the reparametrization of f by m does not have an obvious "visual meaning".

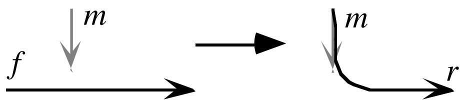

Figure 6: A pathological case (m is perpendicular to f) and how it would be interpreted.

Let d be a distance between two points (or its approximation: $d(f(t),m(k))=(f_x(t)-m_x(k))^2+(f_y(t)-m_y(k))^2$). For each (u,v) in $[0..L_f]^2$, we define the distance D between f_{uv} (the restriction of f to $[u..v]$) and m as the infinitesimal sum of the distances from each point of f_{uv} to its corresponding point on m:

$$D(f_{uv},m) = \int_0^1 d(f(t(v-u)+u), m(tL_m))dt \qquad \text{(a)}$$

The "visually closest" portion of f to m is f_{SE} such that $\forall\ (u, v) \in [0..L_f]^2$, $D(f_{SE}, m) \leq D(f_{uv}, m)$. Note that the length of f_{SE} may not bear any particular relationship to L_m. Performing equal-length matching would prevent most important reparametrization effects, such as adding fine details to a rough outline. The target curve for reparametrization is f in F for which there is a couple (S,E) that minimizes this distance over all curves of F. In fact, we keep as possible candidates all the curves below a certain threshold to allow the user to reparametrize close curves by picking the desired target.

When f and m are discretized into polylines, we can find the parameters S and E that minimize this distance in time directly proportional to the number of segments in F. To do this, we first compute this distance by supposing that $S=E$. Formula (a) on an interval reduced to a single point M becomes a positive polynomial of degree two (formula (b)) that can be minimized in constant time for each segment of f:

$$\underset{t}{Min}\ f_x^2(t) + f_y^2(t) - 2\left(f_x(t)\frac{\int_0^{Lm} m_x(k)dk}{Lm} + f_y(t)\frac{\int_0^{Lm} m_y(k)dk}{Lm}\right) + \frac{\int_0^{Lm} m_x^2(k) + m_y^2(k)dk}{Lm} \qquad \text{(b)}$$

The resulting points M (one for each local minimum of (b)) lie somewhere inside portions (pairs of parameters (S,E)) of the curve that locally minimize formula (a) (figure 7). These points help us find local minima by using a standard minimization technique such as bisection: starting from M at each local minimum of (b), we pose $S=M\pm L_m/2$ and $E=S\pm L_m$, and then iteratively increase or decrease E and S according to the derivative of formula (a), until we find the true local minimum.

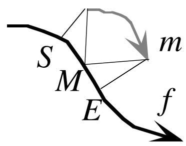

Figure 7: the point M that minimizes formula (b) lies inside $[S..E]$ that minimize formula (a).

This minimization must however take care of potential loops in the target path, which may sometimes prevent finding the true local minima. Currently, we check $[M-L_m..M+L_m]$, to ensure that "natural" cases work properly. To be mathematically correct, we need a loop detection algorithm that works in real time, if such an algorithm can be found.

This technique allows for real time target selection on an HP9000/755 with several thousands of segments (using a fraction of the computing power). Further optimization can be done by using quadratic patches instead of converting each spline into a sequence of segments before target selection, as we currently do. This would greatly reduce the number of patches to test in most cases. Cubic patches would require minimizing polynomials of degree 6: this cannot be done algebraically in the general case and therefore should not lead to significant acceleration.

Reparametrizing the target

Given f, S, E and m, the reparametrization consists of transforming each point of f_{SE} into its corresponding point on m, and making some transformation on f_{0S} and f_{ELf} to preserve the unaffected portions of f while still providing a "harmonious" transition between f_{0S}, m and f_{ELf}. "Harmonious" is another subjective term with several possible definitions. At first glance, the most satisfying may be that the transformation should keep the number and relative positions of the inflection points of f on $(f_{0S} \cup f_{ELf})$. However, this is not always possible. Furthermore, the interaction technique makes such a consideration quite secondary: it encourages either large but imprecise strokes, or fast and small reparametrizing strokes which start and end on the curve, pulling step by step the curve towards the desired shape (figure 8). This way, transitions are always quite small and no inflection points are introduced, except when desired.

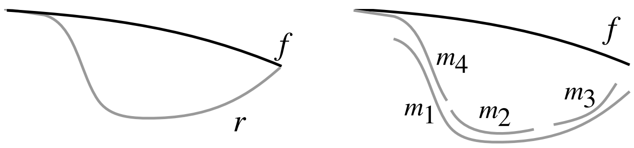

Figure 8: A large modification is harder to perform with a single precise stroke: it is better to begin with a large rough stroke (m_1) and successive small and fast corrections (m_2, m_3 and m_4), each providing finer control over a small portion of the curve.

The method we currently use is quite simple to implement, though it can introduce unnecessary inflection points. First, f and m are converted into polylines; the resulting curve r will be a polyline that will be smoothed using the curve-fitting algorithm of "create" mode. Reparametrization does not thus increase the degree of complexity of the curves. We use a constant k (currently 3) to represent the "smoothness" of the transition and the "transition polynomial" $P(x) = -2x^3+3x^2$. Other functions can be used to introduce a bias toward m or f or to make special transition effects, as long as $P(0)=0$ and $P(1)=1$ to insure the continuity of r. When $S < E$, we start the reparametrization of f at the parameter $S' = S - k\ d(f(S), m(0))$. If S' is negative, then m is said to be an extending stroke: r will start at $m(0)$ and we skip the start transition from f to m. Otherwise (figure 9), we define the line segment m_S, extension of m of length $(S - S')$, of slope $m'(0)$ and such that $m_S(S-S') = m(0)$.

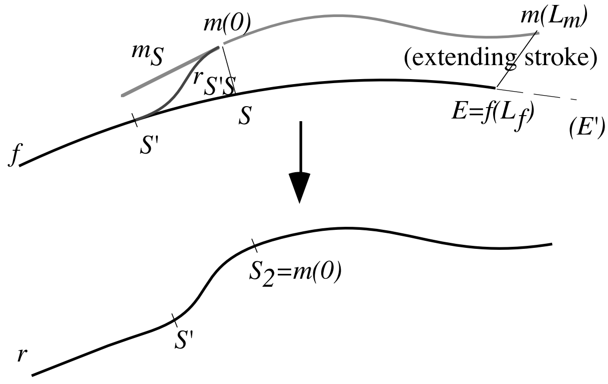

Figure 9: Reparametrization example. *m* and *f* are artificially distant to clarify the figure. Usually, *m(0)* and *f(S)* are much closer, and the inflection point between *S'* and S_2 does not appear.

We then interpolate m_S with each point of f in $[S'..S]$:

```
for t in [S'..S] do
       coeff = P((t-S')/(S-S'));
       r(t) = (1-coeff)*f(t) + coeff*mS(t-S')
done
```

The interpolation using the values of P in [0..1] preserves the C^2 continuity of r where f and m are already C^2. Once this transition is done, f_{SE} is replaced by m. Conversely, to make the transition back to f at the end of m, we set the end of the reparametrization at $E' = E + k\ d(f(E), m(L_m))$. If $E' < L_f$, we interpolate each point of $f_{EE'}$ with m_E, extension of m; otherwise (figure 9) r will end at $m(L_m)$.

In practice, this algorithm has proven to be satisfying, although the transition does not always look smooth enough when f_{SE} and m are too distant. One possibility would be to parametrize the smoothness of the transition, using additional information provided by the stroke, such as variations of pressure on the pen.

A FIRST EVALUATION

As explained in [3], shape-matching tasks are useful for evaluating an interaction technique for spline editing. We informally evaluated the task of changing the representation of an ampersand in Times Roman typeface to Helvetica Roman typeface (figure 10). This task matches that of typeface designers. In specialized software such as FontStudio®, the "&" is represented by 3 paths and about 60 control points. Performing the task is cumbersome and requires removing and adding control points (specially to remove the serifs) in the middle of the curve. Trained users would rather erase the whole drawing and restart from scratch, which deviates from the original shape-matching task. It is out of the question to perform this task without prior experience in manipulating Bézier curves.

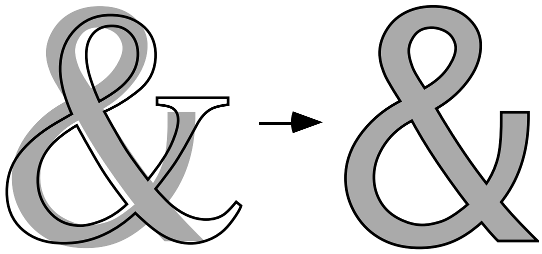

Figure 10: The shape-matching task. The gray "&" is a background image that serves as a template.

With our technique, the task becomes very simple: it consists of drawing over the contour of the Helvetica symbol. All of the following screen dumps are scaled down to 20% of their original sizes. Feedback from the reparametrizing strokes is shown in gray. The user first removes the serifs with three initial strokes that reparametrize the whole end of the symbol (figure 11). Small strokes are then used for precise adjustment of the contour to the target.

***
the figures are not available yet due to postscript errors
when printing them from inside the document
on a Apple Laserwriter
***

Figure 11: The first three strokes to adjust the serifs

The exterior contour of the symbol is then approximated with a few (5) large strokes which follow the contours of the template. Further strokes adjust the curve at a local level (figure 12).

Figure 12: Adjusting the exterior contour

The interior paths are progressively adjusted in 5 large strokes (figure 13). (4 strokes readjusting the serif in between are not shown)

Figure 13: Reparametrizing the interior paths

More strokes (10 ± 4) precisely adjust the portions that did not fit closely, to obtain the final result (figure 14).

Figure 14: Final result with our current algorithms

Higher precision could be obtained by zooming in on the portions that are not perfectly aligned and correcting the paths with small adjustments. However, our current testbed does not allow zooming-in. Because the hand cannot perform efficient adjustments of fewer than two pixels, the current resolution is about 40dpi. Overall, the time to make a perfect match is currently only slightly faster with our technique than it is for a trained user to draw the Helvetica symbol from scratch using Fontographer®. However, a good approximation is obtained in a few fast strokes by even a poor sketcher. We consider that graphic designers, our target users, are less constrained by such limits because they do not usually need to reach a level of precision higher than what they can see and control with their hands.

More importantly, allowing global transformations with one single action allows the designer to proceed with rough sketches and small refinements, rather than choosing the right control-points and editing them one by one to match the desired shape. The directness and transparency of the interaction technique allows the user to focus on the overall shape desired, rather than on low-level considerations such as trial-and-error discovery of the parameters for each control-point.

CONCLUSION AND FUTURE WORK

We have introduced a new way to edit spline curves, using a mark-based interaction technique that allows direct reparametrization instead of modification of individual control-points. We implemented efficient algorithms to perform target identification and reparametrization in real time. Preliminary evaluations show that the technique provides the following advantages:

• *Natural interaction*: Based on observation of paper-based editing by graphic designers, this technique provides the basis for a more natural conceptual model of spline editing.

• *Fast interaction*: A good approximation of the desired shape can be obtained quickly and then refined at an arbitrary level of precision. This supports a top-down approach based on the iterative refinement of drafts and quick sketches rather than progressive construction (by specifying control-points) of a difficult-to-edit final drawing.

• *Direct interaction*: The mark-based interaction technique allows complex reparametrizing data to be entered in one single action instead of separate adjustment of control points.

We plan to pursue several extensions to this work:

• *User evaluation:* The preliminary evaluation was done informally by the author. We found that the proposed shape-matching task is not the most effective way to test this technique, as we cannot strictly compare spline editing using our technique with spline re-creation using conventional methods. We are undertaking formal evaluation procedures, both with "computer-naive" and "computer-aware" graphic designers. This should help us refine the algorithms, especially the reparametrization algorithm, to better match the designers' conceptual models.

• *Use of pressure*: Drawing applications such as Painter® or [13] use pressure information to augment strokes with width or density information. Currently, we know of no interaction technique that allows the reparametrization of the width or density of a curve together with its shape. Our algorithms do not currently use such information either, but the same techniques could be adapted to use pressure data. Obviously, further user studies are needed to make an informed choice.

• *Interaction model*: So far we have focused on the interaction technique and we have not yet worked on the articulation between the different modes within an interaction model. We plan to work on automatic mode switching, feedback of current mode, and integration of graphical attribute editing (e.g., color). This would complete our interaction technique to create a novel interaction style for free-hand drawing.

ACKNOWLEDGMENTS

This work was inspired by conversations with Jean-Daniel Fekete and Francis Freiz. Laurent Mercier and Agnès Rousseaux made many fruitful comments about the pitfalls of current drawing tools and on the art of sketching. Michel Beaudouin-Lafon, Christophe Tronche and Mountaz Zizi provided support and advice on math. Wendy Mackay, Stéphane Chatty and the anonymous reviewers provided many useful comments. This work is partially supported by the "PRC Communication Homme-Machine" research programme.

REFERENCES

1. Banks, M.J. and Cohen, E. *Realtime Spline Curves from Interactively Sketched Data*. in *Computer Graphics* 24 (2), Proceedings of SIGGRAPH'90, ACM. p. 99-107. 1990.
2. Barsky, B.A. *The Beta-Spline: A local representation based on shape parameters and fundamental geometric measures*, PhD thesis, University of Utah. 1981.
3. Bartels, R.H., Beatty, J.C., Booth, K., S., Bosch, E.G. and Jolicoeur, P., *Experimental Comparison of Splines using the Shape-Matching Paradigm,* ACM Transactions on Graphics. Vol. 12 (3), p. 179-208. 1993.
4. Baudel, T. and Beaudouin-Lafon, M., *CHARADE: Remote Control of Objects using Free-Hand Gestures,* Communications of the ACM. Vol. 36 (7), p. 28-35. 1993.
5. Bézier, P., *Emploi des Machines à Commande Numérique*. 1970 , Paris: Masson.
6. Bier, E., Stone, M., Buxton, W. and De Rose, T. *Toolglass and Magic Lenses : the See-Through Interface*. in *Computer Graphics*, Vol. 27 (2), Proceedings of SIGGRAPH'93, pp. 73-80. ACM. 1993.
7. Bier, E., Stone, M., Fishkin, K., Buxton, W. and Baudel, T. *A Taxonomy of See-Through Tools*. in Proceedings of SIGCHI'94, ACM. pp. 358-364. 1994.
8. Carr, R., *The Point of the Pen,* Byte. Vol. 16 (2), p. 211. 1991.
9. Fekete, J.-D., *Tic-tac-toon reference manual*. Images 2001 SA, Antony, France. 1994.
10. Fowler, B. and Bartels, R., *Constraint-Based Curve Matching,* IEEE Computer Graphics & Applications, p. 43-49. September 1993.
11. Kurtenbach, G. and Buxton, W., *GEdit: a testbed for editing by continuous gesture,* SIGCHI Bulletin. Vol. 23 (2), p. 22-26. 1990.
12. Morrel-Samuels, P., *Clarifying the distinction between gestural and lexical commands,* International Journal of Man-Machine Studies. Vol. 32, p. 581-590. 1990.
13. Pudet, T., *Real Time Fitting of Pressure Brushstrokes*. research report 29, Paris Research Laboratory, Digital Equipment Corporation. 1993.

14. Riesenfield, R.F. *Applications of B-Spline approximation to geometric problems of computer aided design*, PhD thesis, University of Syracuse, N.Y. 1973.
15. Rubine, D. *The Automatic Recognition of Gestures*, PhD thesis, Carnegie Mellon University. 1991.
16. Schneider, P.H., *An Algorithm for Automatically Fitting Digitized Curves,* in *Graphics Gems,* A. Glassner, Editor. 1990, Academic Press. A more complete reference, though less accessible is: *Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves*. Master's thesis, University of Washington, 1988.