

# Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes

*James Arvo*  
Computer Science 256-80  
California Institute of Technology  
Pasadena, CA 91125  
arvo@cs.caltech.edu

*Kevin Novins*  
Department of Computer Science  
University of Otago  
Dunedin, New Zealand  
novins@cs.otago.ac.nz

## ABSTRACT

We describe a new sketching interface in which shape recognition and morphing are tightly coupled. Raw input strokes are continuously morphed into ideal geometric shapes, even before the pen is lifted. By means of smooth and continual shape transformations the user is apprised of recognition progress and the appearance of the final shape, yet always retains a sense of control over the process. At each time  $t$  the system uses the trajectory traced out thus far by the pen coupled with the current appearance of the time-varying shape to classify the sketch as one of several pre-defined basic shapes. The recognition operation is performed using shape-specific fits based on least-squares or relaxation, which are continuously updated as the user draws. We describe the time-dependent transformation of the sketch, beginning with the raw pen trajectory, using a family of first-order ordinary differential equations that depend on time and the current shape of the sketch. Using this formalism, we describe several possible behaviors that result by varying the relative significance of new and old portions of a stroke, changing the “viscosity” of the morph, and enforcing different end conditions. A preliminary user study suggests that the new interface is particularly effective for rapidly constructing diagrams consisting of simple shapes.

**KEYWORDS:** Sketching, recognition, morphing.

## INTRODUCTION

The overwhelming majority of computer interfaces in use today employ a pointing device such as a pen or mouse solely for selection and/or for dragging; free-hand input, if used at all, is generally limited to some form of painting or mark-up, where the somewhat arbitrary geometry of the strokes is preserved verbatim. The most common exception to this rule is hand-printed symbolic information, which is accepted by some interfaces in the form of standard typography [17, 12], and by others in the form of abbreviated

“graffiti” [9]. Other exceptions include the use of gesture-based editing [10] and image retrieval [3] commands, and techniques such as the “ARCBALL” method of controlling 3D viewing parameters [11], all of which operate on unconstrained motions made with a traditional mouse or other pointing device. Of course, another important class of exceptions are the interfaces that accept and interpret “sketches” made by the user.

Ever since Ivan Sutherland’s seminal “SKETCHPAD” system was demonstrated in the early 60’s [15], sketch- or pen-based interaction has been recognized as a concise and highly effective means of constructing geometric shapes and specifying constraints. Recent applications of these ideas in the area of computer-aided design include the “SketchIT” system of Stahovich [13], the “SKETCH” system of Zeleznik et al. [18], the projective sketch system of Tolba et al. [16], and the “Teddy” system of Igarashi et al. [8] We submit that there are two fundamental reasons why sketching interfaces have not become more commonplace: 1) it is difficult to reliably interpret free-hand input, and 2) the exact semantics of unconstrained gestural commands are generally not as self-evident as buttons and menus. A result of the first difficulty is that a gesture may be misinterpreted, making it necessary to provide mechanisms for error correction [12]. A result of the second difficulty is that the user may be unclear about which gestures are meaningful or how they are interpreted, even after the fact.

In this paper we present a new paradigm for sketching and describe our initial explorations using a simple prototype system. The motivation behind this work is to lessen the inherent difficulties discussed above by imbuing a sketching interface with immediate and useful feedback. Our approach is to tightly couple gesture recognition with rapid morphing to provide a new and subjectively different form of feedback; here, the feedback is provided *as each pen stroke is drawn*, so that immediate adjustments can be made if the result is not what was intended. This type of feedback also allows sketches to be sloppier and less complete in some instances, since a sketch has served its purpose as soon as it has been recognized for what it is. This frequently requires *less* accuracy than anticipated, a fact rarely disclosed by a typical

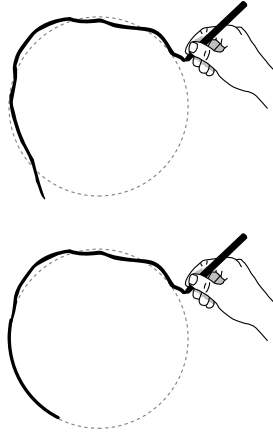


Figure 1: As a figure is drawn, the least-squares match shown by the dashed line is continuously updated. With fluid sketching enabled, the sketch begins to morph toward the most likely figure before the pen is even lifted, as shown on the bottom.

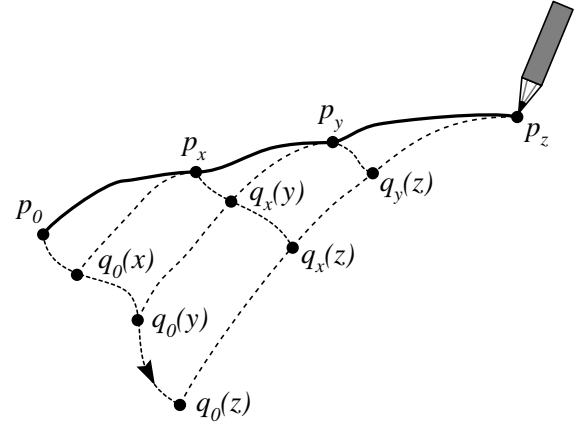


Figure 2: Points  $p_0$ ,  $p_x$ ,  $p_y$ , and  $p_z$  are reached along the trajectory of the pen tip at times  $0 < x < y < z$ , respectively. The function  $q_s(t)$  gives the path of a particular input point if  $s$  is held constant, and the current state of the morphing curve when  $t$  is held constant.

recognition algorithm.

The idea of coupling recognition with instantaneous feedback is not new in itself. For example, in Rubine’s “eager recognition” system [10], an action is triggered as soon as a gesture is uniquely identified. When a rectangle gesture is recognized, a rectangle is drawn in place. Further pointer movement drags one of the rectangle corners. A shape can thus be drawn and modified with a single stroke. While eager recognition helps to reduce the time spent in tool selection, it has no provision for correcting recognition errors. Dynamic feedback is also exhibited by Mortensen and Barrett’s interactive image segmentation tool known as “intelligent scissors.” In this system the strongest boundary between an anchor point and the current pointer position is dynamically updated as the user moves the cutting tool. In contrast to our system, neither of these approaches incorporates morphing to maintain visual continuity.

### AN EQUATION FOR FLUID SKETCHING

In this section we formulate a family of ordinary differential equations (ODEs) that determine how a user-drawn shape changes with time as a result of continuous recognition and morphing; this formalism completely and succinctly expresses the notion of fluid sketching. We refer to the parametric ODE as the *fluid sketching equation*.

Let  $p_t \in \mathbb{R}^2$  denote the two-dimensional coordinates of the pen at time  $t$ . Without loss of generality, we may assume that the initial point of a user-drawn stroke is assigned the time  $t = 0$ , and that the pen is lifted at time  $t = T$ . At some time after a point is drawn, at time  $s > t$ , the point  $p_t$  begins to migrate toward its corresponding position on an *ideal curve*. By an ideal curve, we mean the geometrically precise version of the curve that the user is apparently attempting to draw.

Let  $q_s : \mathbb{R} \rightarrow \mathbb{R}^2$  denote the time-parameterized trajectory of each point drawn by the user; that is,  $q_s(t)$  is the new position of the point  $p_s$  at time  $t$ , which is defined for all  $0 \leq s \leq T$  and  $s \leq t$ . By definition,  $q_s(s) = p_s$  for all  $0 \leq s \leq T$ . We further require that  $q_s(\cdot)$  be a continuous curve, so there are no instantaneous jumps, although it needn’t be smooth. Thus, the graph of  $q_s(\cdot)$ , with  $s$  fixed, is the trajectory of the single point  $p_s$  over time, while the graph of  $q_s(t)$ , for  $0 \leq s \leq t$  and fixed  $t$ , is the shape of the entire morphing curve at time  $t$ . The function  $q$  thus completely specifies the appearance of the curve as it morphs over time, as shown in Figure 2. While there are infinitely many functions  $q$  that may reasonably be associated with the same hand-drawn shape, we shall construct a governing equation for  $q$  that captures its general properties and conveniently characterizes its behavior in terms of simpler functions, each with an intuitive meaning. This single equation will accommodate a wide range of qualitatively different morphing behaviors.

To formulate a governing equation for  $q$  we first introduce some notation. Let  $P : \mathbb{R} \rightarrow \mathcal{P}(\mathbb{R} \times \mathbb{R}^2)$  denote the graph<sup>1</sup> of the user’s pen trajectory as a function of time, and let  $Q : \mathbb{R} \rightarrow \mathcal{P}(\mathbb{R} \times \mathbb{R}^2)$  denote the graph of the morphing curve as a function of time, where  $\mathcal{P}(S)$  denotes the power set of  $S$ . In terms of the functions  $p$  and  $q$ , we may define graphs  $P$  and  $Q$  as

$$\begin{aligned} P(t) &= \{(s, p_s) \mid 0 \leq s \leq t\} \subset \mathbb{R} \times \mathbb{R}^2 \\ Q(t) &= \{(s, q_s(t)) \mid 0 \leq s \leq t\} \subset \mathbb{R} \times \mathbb{R}^2. \end{aligned}$$

One or both of these graphs may influence the trajectories

<sup>1</sup>Here the word *graph* means a set of ordered pairs associating each point in the domain of a function with its image, not a collection of vertices and edges that one studies in graph theory.

followed by each of the points along the morphing curve. It is this feedback of  $q$  influencing its own subsequent time evolution that leads to modeling  $q$  as a differential equation. Specifically, we may express the function  $q_s$  as the solution to the equation

$$\dot{q}_s(t) = f(q_s(t), \Pi[P(t), Q(t)], t - s), \quad (1)$$

which is a one-parameter family of time-dependent first-order ordinary differential equations, where  $\dot{q}_s$  is the time derivative of  $q_s$ , and  $s$  is a free parameter. Here  $f$  determines the instantaneous motion of each point as a function of three arguments: the current position of the given point  $q_s(t)$ , the result of the *matching function*  $\Pi$  applied to both the original stroke and the current shape of the curve, and the “age” of the given point, which is the length of time since it was drawn. As we shall see, there is great latitude in selecting the functions  $f$  and  $\Pi$ , yet Equation (1) clearly identifies the factors that are permitted to influence the motion of each point on the curve. Equation (1) is the fluid sketching equation.

The function  $\Pi$  in Equation (1) incorporates the task of recognition; at each time  $t$ , its role is to identify the corresponding ideal curve, which may involve a variety of time-dependent weighting functions that deem some portions of the curve more important than others. For example, newly drawn portions of the curve may carry greater influence than older portions. The function  $f$ , on the other hand, incorporates the task of locating the point on the ideal curve that corresponds to any given point; the resulting instantaneous motion computed by  $f$  may also be influenced by additional time-dependent weighting functions; for example, a point may gain inertia over time and eventually become immovable.

In this formulation of the evolution equations, the time evolution of  $q$  can be influenced by two distinct factors. First, and most importantly, it is influenced by the continual evolution of the user-drawn curve  $p$ , as the motion of the pen lengthens the curve. This action adds new points to the morphing curve (as is evident by the definition of  $Q$ ) and may *also* change the decision made by the recognizer; this latter event generally results in continual small adjustments to the parameters describing the recognized object, but can potentially change the target object as well, suddenly changing the direction in which each point on the morphing curve is moving. Second, the matching function may be based on the evolving curve itself, which amplifies the feedback; thus, both recognition and nearest-point computations may be influenced by the globally evolving shapes.

## RECOGNITION OF SHAPES

An essential element of the proposed algorithm is the *recognition* stage, which attempts to classify the user-drawn stroke as a fragment of a known geometric figure. In our prototype system, these figures are currently limited to circles, axis-aligned boxes, and line segments, all of which are drawn as

single strokes. Each shape must be equipped with a means of estimating the parameters of the best fit to each user-drawn stroke, and an absolute measure of how good this best approximation is. The system uses these estimates to continuously select the most likely shape and the optimal parameters (e.g. center and radius of a circle) that attains the best match. To make the estimates of quality of fit directly comparable across different shapes, we consistently employ the same metric to compute the distance from a given stroke to any given shape. This metric is the sum of the squares of the distances from each sampled point on the curve to the closest-matching shape.

Finding the best match within a family of shapes cannot be handled by a single approach, however. Even circles and squares require very different strategies. In the case of circles, it is relatively straightforward to express the problem using linear least squares. Given points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  along a user-drawn stroke, we form the over-constrained system of equations

$$\begin{bmatrix} 2x_1 & 2y_1 & 1 \\ 2x_2 & 2y_2 & 1 \\ \vdots & \vdots & \vdots \\ 2x_n & 2y_n & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ \vdots \\ x_n^2 + y_n^2 \end{bmatrix}, \quad (2)$$

and compute a least-squares solution by solving the corresponding normal equations [14]. That is, if Equation (2) is denoted by  $Ax = b$ , the least squares solution is obtained by solving the system  $A^T Ax = A^T b$ . Weighted least squares can be performed using a diagonal weighting matrix  $W$ , with the weight of sample  $i$  appearing as the  $i$ th diagonal entry. In this case the normal equations become  $A^T W^T W Ax = A^T W^T W b$ . In either case, the best-fit circle is then centered at the point  $(c_1, c_2)$  with radius

$$r = \sqrt{c_3 + c_1^2 + c_2^2}. \quad (3)$$

Finding the least-squares fit for a box is more involved, as a box does not admit a simple algebraic expression for its distance from a point. However, we can find the optimal box by means of relaxation, beginning from the smallest box containing the entire stroke. At each iteration we update the shape by applying “spring” forces on the faces of the box resulting from the discrete sample points along the user-drawn stroke. Each point exerts a spring force, pulling the nearest face or vertex of the box toward it. Using gradient descent, we can rapidly converge to a configuration in which the net force on all faces is nearly zero, which corresponds to the optimal box. By damping the spring forces, the optimal box can be found with very few iterations; generally less than 20. This computation is actually less demanding than the corresponding computation for circles, and has the advantage of being inherently incremental. That is, as the stroke changes, we can maintain the optimal box by applying only a small number of relaxation steps.

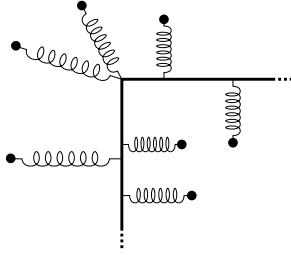


Figure 3: Several points pull on the faces of the box, at the nearest point, with spring forces.

The two examples that we describe here – circles and boxes – are representative of two fundamentally different classes of objects. Circles and other algebraic curves (e.g. lines and higher-order polynomials) all admit straightforward least-squares approximations. Boxes and more general  $n$ -gons, on the other hand, require iterative methods that can accommodate non-smooth approximations. These two different approaches can be mixed without difficulty, since the quality of fit is assessed in the same way in each case, after the optimal shape from each class is found.

### IMPLEMENTATION

In this section we describe very straightforward definitions for the functions  $f$  and  $\Pi$  in Equation (1), and describe the resulting numerical solution. By far the simplest strategy is to define  $\Pi[S_1, S_2]$  to be the unweighted least-squares fit to either the point-set  $S_1$  or the point-set  $S_2$ , without regard to the times associated with the points; that is, by ignoring the time coordinates of the graphs  $S_1$  and  $S_2$ . In the former case, the ideal shape is always selected with respect to exactly what the user has drawn. In the latter case, the ideal shape is always selected with respect to what is currently visible on the screen; that is, the morphing curve. Neither of these pure strategies is completely satisfactory, however. In the former case lines which are no longer visible nevertheless continue to exert an influence over the evolving curve. In the latter case it may be difficult to correct a recognition error; that is, an ambiguous sketch may begin to morph toward an incorrect shape, which increases the likelihood that it will continue to be recognized as the incorrect shape.

A compromise that we have found to be quite effective is to base the matching entirely on the user-drawn strokes, but to weight the recent portion of a stroke more heavily. This approach gradually diminishes the effects of the hidden strokes while placing more emphasis on the portion of the stroke that has changed the least, since it has had a shorter time to undergo a transformation. All of the figures in this paper have been generated using this strategy.

Recall that the role of the function  $f$  is to determine the direction in which a point on the morphing curve should migrate, as well as the speed of its motion. For each point on

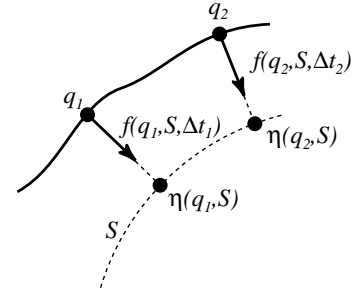


Figure 5: The instantaneous motion of points  $q_i$  toward the points  $\eta(q_i, S)$ , the closest point on shape  $S$ , which is generally also changing with time. The path followed by each  $q_i$  is the solution of an ordinary differential equation defined by the function  $f$ .

the curve, the function  $f$  must base this computation on the current optimal shape, as determined by the  $\Pi$  function, and the time that has elapsed since the point was drawn. The simplest strategy for  $f$  is to simply move in the direction of the closest point on the ideal shape, which we denote by  $\eta(q, S)$ , basing the speed on the distance to this point and a parameter that we call the *viscosity*. In particular, we set

$$f(q, S, t) = \frac{\eta(q, S) - q}{\nu}, \quad (4)$$

where  $\nu \geq 0$  is the viscosity. See Figure 5. A value of  $\nu = 0$  is interpreted to mean instantaneous snapping to the shape  $S$ , while  $\nu = \infty$  means that the user-drawn strokes are retained verbatim.

Note that the viscosity  $\nu$  needn't be constant along the curve; at each point it may be a function of its age  $t$  or position. Indeed, by making  $\nu$  a function of the time since it was drawn, we can enforce a very useful boundary condition. For example, by defining

$$\nu(t) = \nu_0 + \frac{1}{e^{ct^2} - 1}, \quad (5)$$

the viscosity is infinite at the position of the pen, but rapidly converges to the constant  $\nu_0$ . This ensures that the most recently drawn portion of the stroke is always in contact with the pen.

According to Equation (4), each point of the curve  $Q(t)$  moves toward the nearest point on the ideal shape with a speed that is proportional to its distance from  $S$ . This creates an impression of a gradual convergence to the final shape, rather than snapping to the shape. We have adopted this strategy in generating all the figures in this paper. In the middle row of Figure 4, we have set  $\nu = 20$ , and in the bottom row we have set  $\nu = 5$ . Figure 6 illustrates a similar scenario, in which a user draws a very rough box. By the time the sketch is done, it has already morphed into a nearly perfect box.

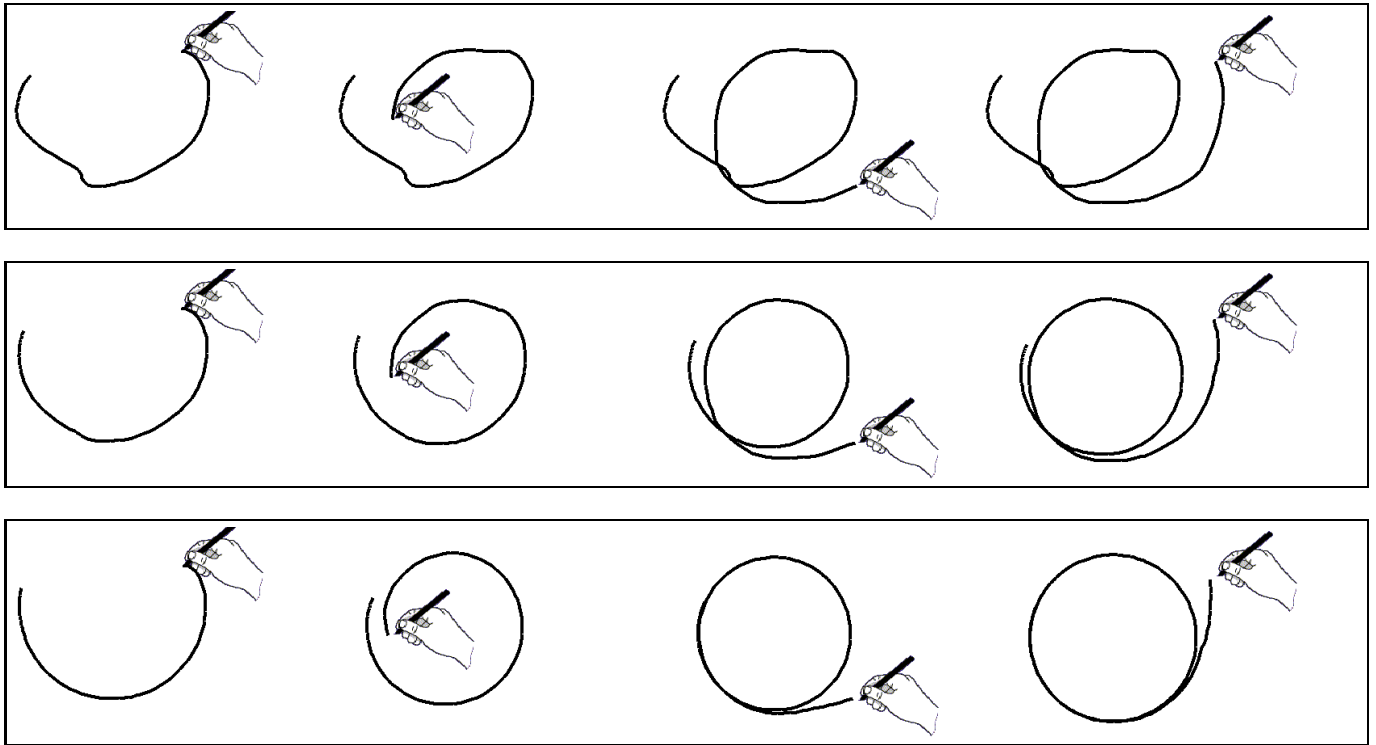


Figure 4: *Top row: a roughly circular pattern is drawn by the user. This data was captured as a user interacted with our prototype system. Middle row: with fluid sketching enabled, the very same trajectory continuously morphs toward the least-squares circle. Bottom row: when the viscosity is decreased, the morphing reaches the current optimal shape more quickly.*

Although the evolving curve  $Q(t)$  is defined mathematically as the solution of a rather messy ordinary differential equation, its numerical solution is completely elementary. Since the evolving shape of the curve is merely intended to create a transient impression, it needn't be computed to high accuracy. Consequently, the solution of the ODE can be carried out using trivial forward Euler steps; that is, by moving each point by its time derivative at each time step. Since all points ultimately converge at the ideal shape, the only inaccuracy that is suffered by using such a crude approximation is in the precise shape of the trajectory followed by each point, and in the location reached on the target shape. Since relatively large errors in both of these aspects of the computation can be tolerated without visibly affecting the outcome, more sophisticated ODE solvers, such as implicit or multi-step methods, do not currently appear to be warranted in this application.

Apart from the numerics of recognition, the most significant complication that arises in implementing fluid sketching is that the morphing must operate concurrently, in a separate thread of execution. This is essential, as the morph may not be complete by the time the user stops drawing. Indeed, many independent threads of execution may conceivably be needed simultaneously to accommodate many shapes that continue to morph as the user proceeds to draw yet an-

other. Unfortunately, multi-threading is not yet a ubiquitous feature of user-interface tool kits [4]. Nevertheless, the multi-threading required for this application can be adequately simulated by periodically inserting morphing updates into the event queue.

### SUBJECTIVE EVALUATION

To begin assessing the utility of fluid sketching we conducted an informal evaluation with eleven subjects, all computer science students who were very familiar with conventional drawing programs, such as Adobe Illustrator and Microsoft Powerpoint. Each subject was asked to reproduce a previously-drawn collection of shapes using our fluid sketching prototype and a similar-looking interface with a tool bar of shapes and a click-and-drag style of interaction. We shall refer to the latter interface as "conventional." The subjects were given unlimited time to familiarize themselves with both interfaces, and were asked to complete a survey after performing the assigned task.

The results of this study suggest that the fluid sketching paradigm is easy to pick up, and can quickly begin to compete with more conventional methods. However, the study also indicated some of the shortcomings of fluid sketching as it is currently implemented.

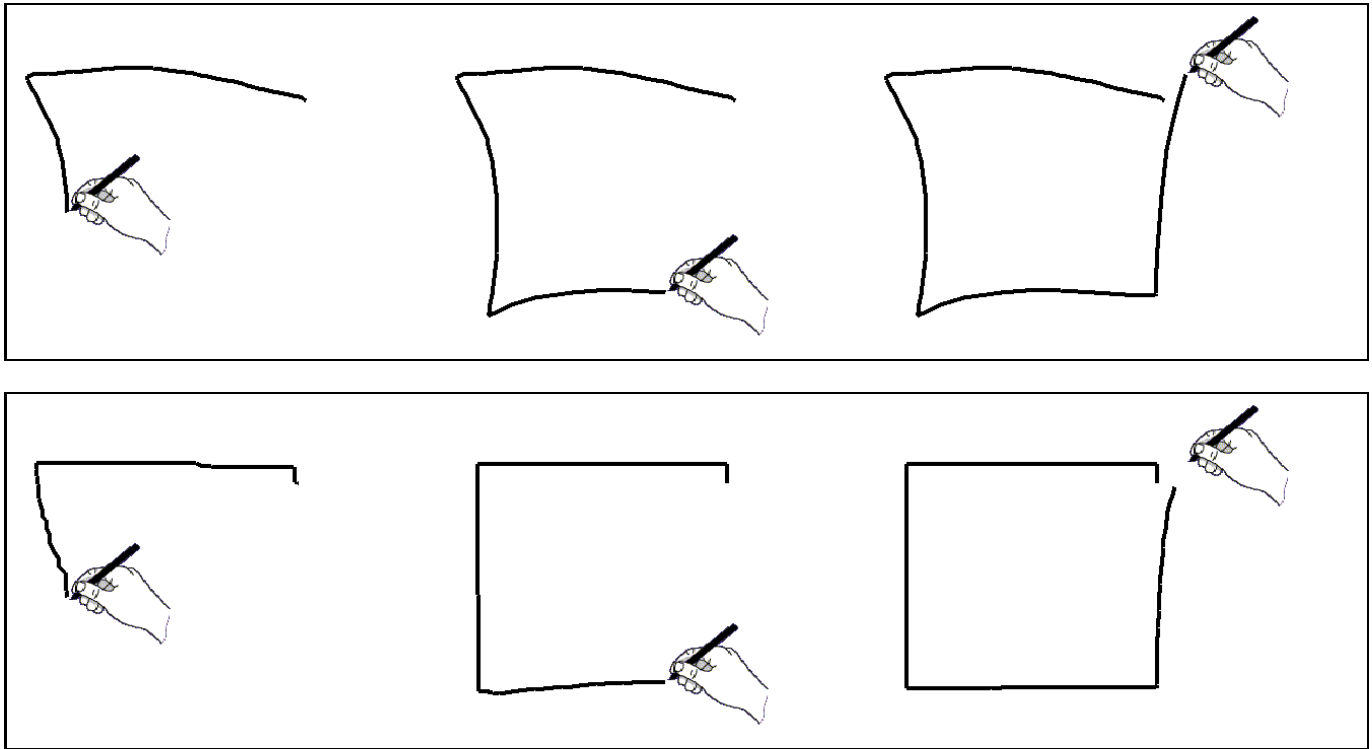


Figure 6: *Top row: the user sketches a rough box. Bottom row: with fluid sketching enabled, the identical user-drawn stroke continuously morphs toward the least-squares box as it is being drawn.*

When asked which interface felt more natural, five subjects were neutral, two slightly favored the conventional interface, and four slightly favored fluid sketching. When asked which interface they preferred for creating accurately placed shapes, most subjects slightly favored the conventional interface. However, when asked about approximately placing shapes, eight subjects favored fluid sketching, with five of them favoring it strongly. Finally, when asked which they preferred for rapidly creating new approximate drawings, nine of the subjects favored fluid sketching, with five of them favoring it strongly. Thus, in situations where accuracy of placement is less important, the majority of users in this study preferred fluid sketching, citing the fact that repeated tool selection was unnecessary, and that it was simply more fun to use. As a further auspicious sign, without prompting all eleven test subjects elected to continue experimenting with the fluid sketching interface after completing the questionnaire.

The study also revealed the major shortcoming of fluid sketching. One subject found that the system did not readily recognize his rectangles. More significantly, nearly all the subjects stated that there must be a means of editing the result after it is drawn; while this capability can be realized within a purely fluid sketching interface, it has not yet been implemented.

## DISCUSSION

This work constitutes an initial exploration of fluid sketches, which tightly couples recognition and morphing to create a continuous and immediate form of feedback. The rationale for fluid sketches is twofold: first, feedback on the recognition of the figure is given immediately, and second, seeing the true geometry of the shape can aid in drawing it. For example, the approximately parallel sides of a box quickly become exactly parallel, making the final size and shape of the box evident and allowing for immediate correction if desired. As a further aid, it is also possible to immediately show the ideal shape corresponding to the hand-drawn stroke, as shown in Figure 1. Thus, one can immediately see the extent of a circle after drawing only a small portion of it. This is currently an option in our system.

The idea of fluid sketching can be taken to different extremes. For example, morphing can be made effectively instantaneous, where the user-drawn stroke immediately snaps to the ideal shape. At the other extreme, the ideal shape is drawn as an overlay, but the original user-drawn stroke remains unaltered. The second extreme provides feedback about the recognition process, but does not change what the user has drawn. Retaining the original input may be important, particularly in the early stages of a design process [5, 16, 8]. In the context of architectural drawings, sloppiness may connote lack of commitment or known inaccuracies [7]. While

we agree that both instant morphing and suppressed morphing are likely to find application, we have observed that middle ground offered by our current system has wide appeal.

One scenario that is unique to fluid sketching is that in which the character of the user-drawn sketch changes mid-way through a stroke, as depicted in Figure 7. Here, the user begins to draw what looks like a slanted line, but then proceeds to draw an arc, and finally a sharp edge indicating a box. In this case the system changes its interpretation twice as the user draws; it first re-morphs the line into a circular arc, and then the entire curve into a portion of a box. Such a scenario may occur as a result of sloppiness on the part of the user, in which case the feedback can be instrumental in correcting the error, or the user may have simply changed her mind, in which case the already-drawn portion of the figure is seamlessly updated, as though the intention had been clear from the start.

Even in its current limited form, the idea of fluid sketching may find application to free-hand on-line construction of simple diagrams, such as finite state machines and Venn diagrams, which have simple and complete semantics [1, 6]. Such diagrams rely upon very simple geometrical elements, and are often far superior to more abstract mathematical notation.

While the fluid sketching approach is feasible for simple geometric shapes, it is unlikely that it can be applied so successfully in the context of character recognition and morphing [2]. For fluid sketching to work, recognition must occur almost instantaneously. However, character recognition often requires more context and more subtle distinctions, which unfortunately precludes recognition in mid-stroke [12].

#### FUTURE WORK

Our current system was developed as a proof-of-concept. A full scale drawing application will require a larger repertoire of shapes. Polynomial curves and general polygonal shapes are high priorities. Expanding the repertoire of shapes involves more than simply adding additional classes of shapes to the recognizer, however. The case of an ellipse illustrates the additional complications nicely. If general ellipses were added, it would then become almost impossible to produce a perfect circle, since the extra degrees of freedom of an ellipse will almost always lead to a better fit. How, then, can one specify a perfect circle by means of an imprecise sketch? There are at least two distinct strategies that one might pursue here. First, the initial recognition can be based on well-separated basic shapes, which are then distorted after the fact. That is, the recognizer might only look for perfect circles, or perhaps ellipses of some minimum eccentricity. Once the shape is drawn, it could be distorted using more conventional pick-and-drag methods. Another possibility is to draw highly-constrained special cases in a distinct manner. For example, an exact circle might be specified by tracing the entire shape twice.

Ultimately, it will be necessary to accommodate multi-stroke shapes, such as arrows. This will require some way of determining whether a collection of strokes belong together, which is similar to the problem of grouping strokes that represent a single hand-written character [12].

Although it would be easy to incorporate editing operations into our system using conventional tools, we would like to explore implementing them in a manner that is more consistent with the spirit of fluid sketching. One possibility would be to use overwriting to influence the least-squares fit of existing shapes.

We look forward to performing user studies to determine the strengths and weaknesses of fluid sketching more precisely. These studies will also help us to select among the many possible strategies embodied by the fluid sketching equation. Moreover, optimal morphing rates and recognizer parameter settings (e.g. least squares weights) will likely be user- and task-specific, which will only become apparent after extensive use.

#### ACKNOWLEDGMENTS

This work was supported by a National Science Foundation Career Award (CCR9876332).

#### REFERENCES

1. James Arvo. Computer aided serendipity: The role of autonomous assistants in problem solving. In *Proceedings of Graphics Interface '99*, pages 183–192, Kingston, Ontario, June 1999.
2. James Arvo and Kevin Novins. Smart text: A synthesis of recognition and morphing. In *AAAI Spring Symposium on Smart Graphics*, pages 140–147, Stanford, California, March 2000.
3. Alberto Del Bimbo and Pietro Pala. Visual image retrieval by elastic matching of user sketches. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):121–132, February 1997.
4. Emden R. Gansner and John H. Reppy. A multi-threaded higher-order user interface toolkit. In Bass and Dewan, editors, *User Interface Software*, chapter 4. John Wiley & Sons, New York, 1993.
5. Mark D. Gross and Ellen Yi-Luen Do. Ambiguous intentions: A paper-like interface for creative design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, pages 183–192, Seattle, Washington, 1996.
6. Eric M. Hammer. *Logic and Visual Information*. Center for the Study of Language and Information, Stanford, California, 1995.
7. Marti Hearst. Trends and controversies: Sketching intelligent systems. *IEEE Intelligent Systems*, 13(3):10–19, May/June 1998.

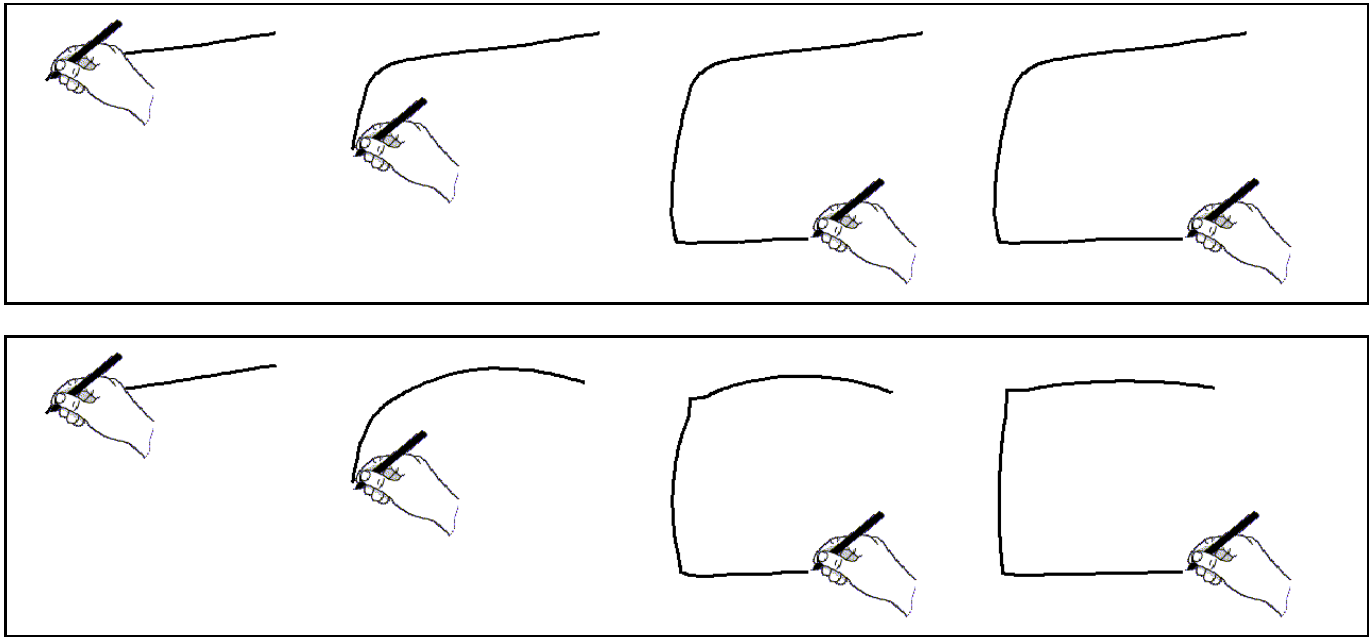


Figure 7: Top row: the user draws a shape that initially looks like a slanted line, then a circular arc, and finally a box. Bottom row: when fluid sketching is enabled, the initial shape morphs toward a line but then changes direction and begins to morph toward a circle. Finally, when the figure is clearly seen to be a box, the system (correctly) changes direction again and begins to morph toward a box. The transformation is nearly complete in the last frame on the right. This feedback continually informs the user of the system's best guess.

8. Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, pages 409–416, August 1999.
9. I. Scott MacKenzie and Shawn X. Zhang. The immediate usability of graffiti. In *Proceedings of Graphics Interface '97*, pages 129–137, Kelowna, B.C., May 1997.
10. Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25(4):329–337, July 1991.
11. Ken Shoemake. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92*, pages 151–156, Vancouver, Canada, 1992.
12. Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Proceedings of Graphics Interface '99*, pages 84–91, Kingston, Ontario, June 1999.
13. Thomas F. Stahovich. *SketchIT: A Sketch Interpretation Tool for Conceptual Mechanical Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.
14. Gilbert Strang. *Linear Algebra and its Applications*. Academic Press, New York, second edition, 1980.
15. Ivan E. Sutherland. *SKETCHPAD: A Man-Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, January 1963.
16. Osama Tolba, Julie Dorsey, and Leonard McMillan. Sketching with projective 2d strokes. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 149–157, Asherville, North Carolina, November 1999.
17. Larry S. Yaeger, Brandyn J. Webb, and Richard F. Lyon. Combining neural networks and context-driven search for online, printed handwriting recognition in the Newton. *AI Magazine*, 19(1):73–89, 1998.
18. Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for gestural modeling. In *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, pages 163–170, August 1996.