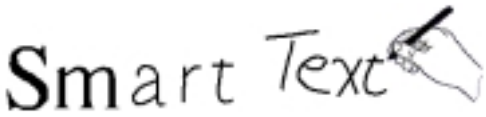


# A Synthesis of Recognition and Morphing

**James Arvo**
California Institute of Technology
arvo@cs.caltech.edu

**Kevin Novins**
University of Otago
novins@cs.otago.ac.nz

**Abstract**

We present a new approach to managing the appearance of handwritten symbolic information for pan-based systems in which text is recognized and gradually transformed into clean typography. The approach is best suited to pen-based symbolic input and manipulation in which the user interacts directly with the display. Once a collection of user-drawn strokes has been recognized as comprising a specific symbol, the strokes begin a gradual metamorphosis into the corresponding raster image from an appropriate font. The transformation retains legibility at all intermediate stages and can proceed smoothly at widely differing rates. The metamorphosis is accomplished by an energy-based stroke correspondence algorithm, followed by level-set shape transformation that gradually introduces characteristic features of a given font, such as serifs and spurs. This technique leads to a qualitatively different form of pen-based interaction in which the user enjoys the benefits of character recognition and highly legible typefaces without abrupt changes in appearance or positioning of the text. The interplay between hand-written symbols and pre-defined raster fonts suggests a number of new methods for correcting character recognition errors; for instance, the recognizer can be given additional hints by "touching up" an incorrectly recognized symbol.

## Introduction

Pen-based computer systems with built-in handwriting recognition are finding an increasing number of applications. Typically such a system will accept a stream of hand-drawn text or "graffiti" (MacKenzie & Zhang 1997) which it then attempts to identify as a sequence of known characters. Once recognized, the system displays the corresponding symbols using a built-in font, replacing the user's original input and typically repositioning characters and words as well. The substitution of a raster font for the original handwritten input serves several purposes. First, it provides feedback to the user on the system's current interpretation of the input, since recognition failures becomes immediately apparent. Second, it provides a more compact and legible representation of the text, which is suitable for other purposes, such as word processing or algebraic manipulation.

One aspect of current pen-based systems that makes them less intuitive than writing on paper is the means by which user input is transformed; often as a sudden transition at the conclusion of a word or phrase. Such a transformation can be distracting, as the correspondence between the hand-written symbols and the currently displayed symbols can be momentarily lost. This is primarily due to instantaneous repositioning of symbols caused by differences in both the sizes and shapes of the symbols drawn by the user and those of the selected typeface. Repositioning can mask recognition errors and destroy the user's mental map of words or symbols on the page. This is particularly troublesome in the context of constructing complex mathematical formulae, in which subtleties in the two-dimensional layout carries a great deal of information (Arvo 1999; Smithies, Novins, & Arvo 1999).

In this paper we present a new approach to managing the appearance of hand-printed pen input in which all transitions are gradual. The metaphor is that of virtual ink that gradually reorganizes itself into clean type, as shown in Figure 10. The primary technical challenge of the new approach is that of performing a visually appealing transition from hand-drawn to computer-drawn text, maintaining legibility and approximate spatial organization at all intermediate steps.

The input to our prototype system consists of a sequence of strokes, organized as a polylines, that represent temporally distinct (although possibly intersecting) segments of the hand-drawn characters. The system then identifies stroke groups as specific symbols and generates a sequence of intermediate shapes for each symbol; the final representation is a high-quality image of each recognized character generated from a standard outline font.

To reconcile the hand-drawn polylines and the final outline font, which are extremely different representations, we introduce an intermediate stroke font that serves as both the end point of an energy-based morph and the starting point for a level-set-based morph. Ac-

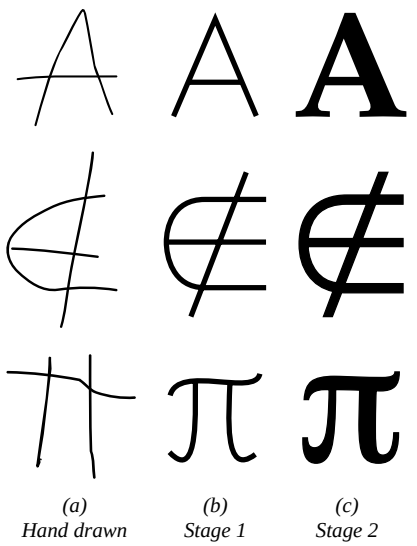


Figure 1: *Each morph has two conceptual stages. In the first stage, lines are blended into lines. In the second stage, lines are expanded to fill in all the font features.*

cordingly, our method proceeds in two stages, as shown in Figure 1. In the first stage, the polyline representation of the input is transformed into the polyline representation of the stroke font. In the second stage the polyline representation of the stroke font is transformed into the polygon representation of the final outline font.

In addition to providing a continuous appearance, gradual metamorphosis from drawn characters to a typeset character offers the user the opportunity to correct recognition errors mid-way through the process; for example, by re-drawing specific features of the character that have receded, or crossing out features that have incorrectly appeared (e.g. a "$\lambda$" may have been incorrectly recognized as an "X"); these steps can be taken before the transformation to the incorrect character is complete, and the strokes still retain much of their original identity.

An alternative to gradual transformation would be "crisp" transformation immediately upon completion of each symbol. This would give instant feedback from the recognizer and confine all geometric changes to the user's immediate focus of attention. Despite these benefits such an approach is not viable, as there is insufficient information to disambiguate multi-stroke symbols until at least part of the following symbol has been drawn, or some other completion cue is given (Smithies, Novins, & Arvo 1999). Furthermore, it is frequently convenient to draw symbols at a different scale or at a different position than the ultimate typeface; gradual transformation allows for alignment of abutting text, superscripts, and subscripts, to be drawn at the larger scale, before transformation to the smaller typeface is complete.

## Previous Work

The most challenging aspect of shape metamorphosis is to maintain important features throughout the transition, so that intermediate stages are recognizable and meaningful (Sederberg & Greenwood 1992; Seitz & Dyer 1996). Currently, there is no fully automatic means of constructing morphs in general; often the user must identify salient features that are to be preserved. Sederberg and Greenwood presented a physically-based approach for automatic morphing of polygonal objects that partially addresses this problem (Sederberg & Greenwood 1992). They seek to minimize deformations that would destroy important features by imposing energy penalties on bending and stretching. Effective transformations among pairs of 2D images requires that features be identified and put into correspondence using constructions such as oriented line segments (Beier & Neely 1992) or a grid points (Wolberg 1982). To partially automate this process, Lee *et al.* used "snakes" to help identify and match image features, which again invoked an energy minimization principle (Lee *et al.* 1995). Energy-base line morphing has also been used as the basis of recognition tasks such as image retrieval (Del Bimbo & Pala 1997), where a small deformation energy indicates a high probability of a match.

Because conventional feature-based morphs interpolate the deformation field between feature points, it is difficult to preserve the integrity of high-contrast features without establishing an inordinate number of feature correspondences. For this reason, feature-based approaches are not currently suitable for morphing text, where high-contrast edges dominate, and intermediate stages must retain these sharp features.

## Recognition

Hand-written symbol recognition is a central component of our system, as it determines the actual content of what the user has written, and thereby also determines the sequence of symbols to morph into. The recognizer is also intimately involved in deciding which collections of strokes constitute characters. This latter task we refer to as *stroke grouping*.

Recognition in our system is performed only at the level of symbols, which are encoded as collections of polylines representing individual user-drawn strokes. We use an on-line recognition algorithm based on nearest-neighbor classification in a feature space of approximately 50 dimensions. That is, each potential character is analyzed in terms of 50 characteristic features, which then serve as coordinates in a Euclidean space. Some of the more basic features include the number of strokes, the mean slope of each stroke, the mean gap length between strokes, and the mean curvature of each stroke. In general, each feature corresponds to a scalar-valued function, or functional, defined on a set of polylines representing hand-drawn strokes. These features are designed to be scale invariant so that the

recognition rate is unaffected by the size at which the user writes.

Many of the feature functionals are implemented by defining a two-dimensional function over the bounding box of the character and summing the values obtained at the vertex positions, or at points that are a given arclength along the segments. Sill others feature functionals are computed by means of histograms of vertex positions, histograms of angles between adjoining segments, and relative positioning of strokes. A similar strategy is described by Avitzur (1992).

Each point in this 50-dimensional feature space represents a collection of similar characters. By training the system on 10 to 20 samples of each character, we obtain points that represent prototypical characters. It is the distance to these points that we use to determine which character the user has drawn. The weight of each feature is symbol-dependent, and is determined by the variability of that feature within the samples supplied for each symbol; high variability down-weights the contribution of that feature.

Nearest-neighbor classification is a simple but extremely fast method for character recognition. Several factors allow this simple approach to achieve comfortably high recognition rates. First, since the recognition task is confined to printed symbols, which typically consist of three or fewer strokes, the number of patterns that must be distinguished is relatively small; on the order of several hundred. Second, since all user input is acquired on-line, segmentation of user input into distinct strokes is immediate, since all strokes, even those that cross, are well separated temporally. Thus, timing information eliminates one of the most troublesome phases of off-line handwritten character recognition.

To train the system, the intended user supplies many hand-written samples of each character. These samples are stored and used to produce both the classification points in feature space as well as the symbol-dependent feature weights, which are simply the standard deviations of the features. Although recognition is theoretically user-dependent, the system is relatively user-independent in practice. For example, even though all of our experiments were performed using training samples supplied by only one of the authors, others had little difficulty in using the system. In part, this is because the mapping from patterns to symbols implemented by the recognizer is many-to-one, meaning that the system can recognize a variety of differing styles for each character. This characteristic accommodates different users as well as different styles employed by a single user.

Strategies for attaining higher recognition rates include the use of neural nets (Yaeger, Webb, & Lyon 1998) and use of context, as described by Miller and Viola (1998). Virtually any recognition module could be incorporated into our system. The only fundamental requirement that is placed on the recognition module is that it must be capable of ranking the $k$ most likely candidates by a numerical measure of confidence, and that these confidence levels be directly comparable among different symbols. The latter requirement is imposed by the stroke grouping algorithm, as described in the following section.

### Stroke Grouping

Grouping of distinct strokes into symbols is an integral part of hand-printed symbol recognition. For this task we use a search technique that lags several strokes behind the user, keeping the strokes in a queue, looking for stroke groupings that result in high-confidence recognition (Smithies, Novins, & Arvo 1999). All possible contiguous groupings of the strokes in the queue are considered and assigned a confidence measure equal to that of the *least* likely symbol in the group. The first symbol of the grouping with highest confidence is retained; then the corresponding strokes are removed from the queue and the grouping process is restarted.

## Shape Transformation

In this section we describe how the user-drawn shape is transformed, or *morphed*, into clean type after recognition. The morph is performed in two stages, which are described below. Because we whish to allow morphs to happen very gradually, it is essential that the text remain legible at all intermediate stages; otherwise, the text would be rendered useless for significant periods of time. Consequently, both stages of the morph are designed to retain maximum legibility at each intermediate stage.

### Stroke and Outline Fonts

The goal of the morphing process is to convert the user's handwritten symbols into clean type. We refer to the final typeset font as the *outline font*, as its characters are represented by polygonal boundaries. We also require an intermediate representation, which we call the *stroke font*. Each character in the stroke font is represented as a set of polylines, or a *multiline*. A stroke character acts as both the goal of the Stage-1 morph and as the starting point of the Stage-2 morph. Our system uses an encapsulated PostScript file generated by Adobe Illustrator to encode both the outline font and the corresponding stroke font which we have carefully designed for this application.

The following properties are desirable properties for a stroke font, and constrain their design:

- *Containment:* Each stroke character is entirely contained within its corresponding outline character.
- *Clarity:* Each character of the stroke font should be aesthetically pleasing as well as easily recognizable.
- *Correspondence:* The strokes comprising each character in the stroke font should correspond as closely as possible to the strokes of a typical hand-drawn rendition.

The *containment* property is a requirement of our Stage-2 morphing algorithm, which uses level sets to

interpolate intermediate shapes between the stroke and outline characters. Both the *clarity* and *correspondence* properties are required to make the Stage-1 morph legible throughout. In practice, correspondence is difficult to achieve simultaneously with containment and clarity. Frequently handwritten symbols differ in topology from their typeset counterparts. For example, a "g" is usually typeset with two loops, but usually written with a loop and a hook.

One approach to generating a stroke font is to use an automatic skeletonization algorithm to create stroke characters directly from the corresponding outline characters. We have implemented such a technique using the *discrete Chordal Axis Transform* (CAT), which was introduced by Prasad (1997). The CAT is based on the observation that midpoints of the edges in a Constrained Delaunay Triangulation (CDT) of a polygonal shape will closely resemble the medial axis of the polygon.

The CAT skeleton automatically meets the containment criterion, and is typically a highly recognizable representation of the outline character it was generated from. However, every skeletonization algorithm is prone to generating shapes with many small jagged branches, especially in or near the small features of an outline font, such as spurs or serifs. Even if these small features are pruned away, the automatically generated skeletons rarely have the aesthetically pleasing form of a well-designed font. An example of a CAT skeleton before and after pruning is shown in Figure 2.

To overcome this difficulty, we rely upon carefully hand-drawn stroke fonts that have been designed to meet the clarity and correspondence properties as closely as possible. A comparison of a hand-designed stroke character to and automatically generated stroke character is shown in Figure 2. Nevertheless, we have found that the CAT skeleton is still a useful tool; in our system, this automatic skeletonization algorithm is used as a temporary measure for outline fonts that do not yet have a corresponding hand-designed stroke font.

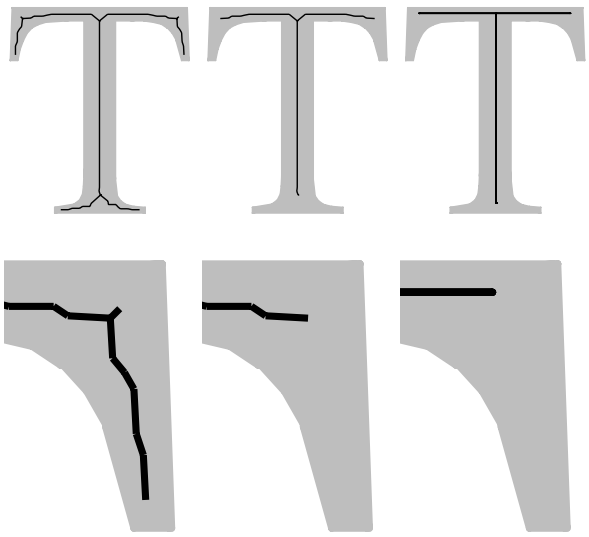

Figure 2: *A skeleton versus a hand-designed stroke character for the symbol "T" (top row) and a close-up of a serif region (bottom row). The raw CAT skeleton on the left is pruned, resulting in the automatically generated stroke character shown at center. On the right, a hand-designed stroke character is shown. (Lines are overlaid on the filled outline character for reference.)*

## Stage-One Metamorphosis

The Stage-1 morph transforms a handwritten symbol into the corresponding character of the stroke font. Both the handwritten symbol and the corresponding stroke characters are represented as multilines. The most important part of the Stage-1 morph is establishing a very natural correspondence between the stokes of the hand-drawn symbol and the stroke character. This must be accomplished quickly and reliably without user intervention. Finding such a correspondence is complicated considerably by the fact that the user input and the corresponding stroke character often consist of a different number of polylines; this is largely due to the variety of writing styles among users, or even employed at different times by the same user. Once this stroke-to-stroke correspondence is established between the *source* and the *target* multilines, the Stage-1 morph can easily perform a gradual transformation of the source strokes into the target strokes using simple linear interpolation of corresponding vertices.

There are two sub-problems associated with finding a suitable correspondence. First, the source and target shapes must be partitioned into an equal number of pieces. Second, the pieces must be put in one-to-one correspondence in the most natural way, so that intermediate shapes retain as many of the features of the source and target shapes as possible.

The obvious units into which to partition the characters into are the individual strokes themselves. If the number of strokes in the source and target characters are equal, and a satisfactory correspondence between the strokes can be found, as described below, then this is the partitioning that is used. If, however, the number of strokes does not match, at least one of the shapes must be altered by either *splitting* or *joining* existing strokes. See Figure 3. By splitting, we mean breaking a single stroke into two stokes at a sharp bend. By joining we mean combining two strokes into one if they are nearly contiguous and collinear line segments.

Once the number of strokes, $n$, in source and target are equal, a combinatorial optimization process is performed to find the best possible matching among the strokes. Since $n$ is always small, with $n = 1$ and $n = 2$ being most typical, it is reasonable to search through all $n!$ possible correspondences, where each line-to-line correspondence is also optimized according to direction; that is, parameterizations in both directions are

*(a) Splitting* *(b) Joining*

Figure 3: *To create a one-to-one correspondence between strokes, the strokes of a character may be (a) split at the sharpest bend, or (b) joined where they are most nearly collinear.*

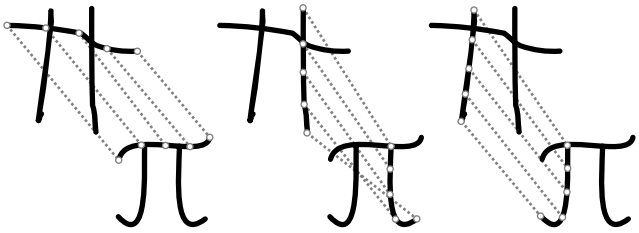

Figure 4: *The "energy" associated with a pair of strokes is estimated using the distances between corresponding points.*

tested, and the resulting in the lowest "energy" is retained. The correspondence that is sought is the the one that minimizes the total "energy" associated with the transformation. The energy is simply computed as the sum of the squares of the distances that points on one stroke must travel to reach the other stroke; no analogue of bending or stretching work are needed. See Figure 4. Alternatively, the line blending algorithm such as the one proposed by Sederberg and Greenwood (1992) could be used to define a more physically meaningful measure of the work needed to change one polyline into another.

The complete algorithm is outlined in pseudo-code in Figure 5, where $|P|$ means the number of strokes in the multiline $P$. Note that the splitting and joining phases are not guaranteed to produce an equal number of strokes in $P$ and $Q$, since both operations are thresholded. When no more sharp bends or sufficiently collinear segments remain, no more splitting or joining is done. The optimization then finds a match for all the strokes of $P$ or all the stokes of $Q$, whichever is fewer. The remaining unpaired strokes are then associ-

FindCorrespondence(Multiline:$P, Q$)

**begin**

*Try to make $|P| = |Q|$ by joining segments.*

**while** $|P| > |Q|$ **and** MostCollinear($P$) $< C_1$ **do**
join the two most collinear strokes of $P$

**while** $|Q| > |P|$ **and** MostCollinear($Q$) $< C_1$ **do**
join the two most collinear strokes of $Q$

*Try to make $|P| = |Q|$ by splitting at bends.*

**while** $|P| < |Q|$ **and** MaxBend($P$) $> C_2$ **do**
break $P$ at the sharpest bend

**while** $|Q| < |P|$ **and** MaxBend($Q$) $> C_2$ **do**
break $Q$ at the sharpest bend

*Search for minimum-energy correspondence among the first* $\min(|P|, |Q|)$ *strokes.*

$E_{\max} \leftarrow \infty$
**for** each permutation $\pi$ of $P$ **do**
$E \leftarrow$ MorphEnergy($\pi P, Q$)
**if** $E < E_{\max}$ **then**
$E_{\max} \leftarrow E$
$\pi_{\max} \leftarrow \pi$
**endif**
**endfor**
$P \leftarrow \pi_{\max} P$

*Force $|P| = |Q|$ by adding degenerate segments.*

**if** $|P| < |Q|$ **then**
add degenerate segments to $P$

**if** $|Q| < |P|$ **then**
add degenerate segments to $Q$
**end**

Figure 5: *Pseudo-code for finding the best correspondence among the strokes of two multilines.*

ated with degenerate strokes in the other multiline; that is, strokes of zero length. This allows any unmatched strokes, such as serifs, to simply grow from nothing out of the source shape, or shrink to nothing in the target shape.

After the one-to-one correspondence is established, the Stage-1 morph is performed by linear interpolation among the polyline pairs. This is done by associating points at proportional arclengths along corresponding polylines, which requires new vertices to be introduced to both polylines.

## Stage-Two Metamorphosis

The Stage-2 morph gradually converts a stroke character into the corresponding outline character, which is potentially embellished with common typographic fea-

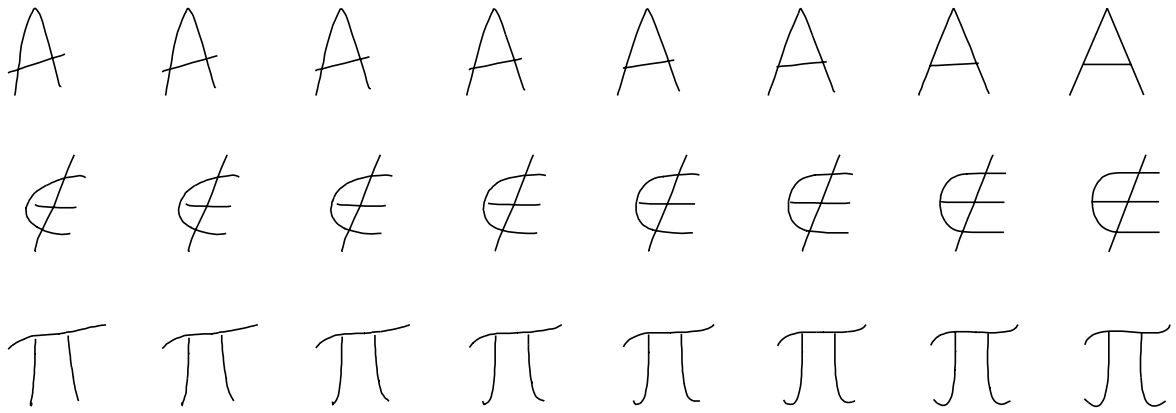

Figure 6: *Three examples of the Stage-1 metamorphosis, which maps user-drawn characters to corresponding characters of a stroke font.*

tures such as serifs, spurs, and variable line weight.

In order for the Stage-2 morph to appear gradual, the typeset character must appear to grow from the stroke character. General-purpose image morphing (Beier & Neely 1992) does not provide fine enough control to give the intended effect. Furthermore, it, as well as other raster-based morphs (Novins & Arvo 1999) require significant precomputation in order to run in real time. This makes their use awkward for our application, since we must be able to perform the morph at varying scales to match the size of the user's strokes.

Our problem is one of morphing from a multiline, or set of polylines, to a set of polygons. To perform such a morph, we define a two-dimensional scalar field in the plane such that points on the stroke font have field value 0 and points on the outline font boundary have field value 1. Let

$$\alpha(x) = \frac{S(x)}{S(x) + O(x)},$$

where $S(x)$ is the distance from the point $x$ to the closest point of the stroke character, and $O(x)$ is the distance from $x$ to the closest point on the outline character. Since $S(x)$ and $D(x)$ are non-negative, the range of $\alpha(x)$ is $[0, 1]$. This range is used to define contours that are any given fraction of the way through the transition from stroke to outline. That is, the level set of all points $x$ with $\alpha(x) \leq t$ will define the intermediate representation of the character at "time $t$." This technique is equivalent to morphing via interpolation of a signed distance function (Turk & O'Brien 1999; Novins & Arvo 1999).

To compute the level sets in real time, we calculate $\alpha(x)$ at the vertices of a triangular mesh and linearly interpolate all intermediate values. The approximate level sets can then be rendered using a 2D equivalent of the Marching Cubes algorithm (Lorensen & Cline 1987) that operates on triangles rather than on cubes.

To create the mesh used in morphing we again resort to the constrained Delaunay triangulation. All vertices of the stroke character are placed in the mesh with an $\alpha$ value of 0 and all vertices of the outline character are placed in the mesh with an $\alpha$ value of 1. The line segments of the stroke and outline polylines are then constrained to be part of the final mesh.

One complication that arises from this approach is that some small fraction of the edges added by the constrained Delaunay triangulation will have both vertices on the stroke character, or both vertices on the outline character. This is a problem since the 2D marching cubes algorithm would cause any triangle whose vertices all have the same $\alpha$ value to "pop" into the morph in a single step, causing an abrupt change in the shape of the character.

Fortunately, this problem has a simple remedy. We simply examine all edges that are added to the mesh by the triangulation algorithm; for any edge that connects two vertices with the same value, we add the midpoint of the edge to the triangulation, along with its true $\alpha$ value. In effect, this algorithm generates a CAT skeleton "on the fly" in regions of the outline character that are not visible to the stroke character. This completely eliminates the problem of popping triangles, making the entire Stage-2 morph very gradual.

Figure 7 shows the process of creating a Stage-2 morph. At left, the raw constrained Delaunay triangulation between stroke and outline characters is shown. Dots mark the midpoints of unconstrained edges that connect two vertices with the same $\alpha$ value. The result of adding these points to the mesh is shown in the center panel. On the right, an interpolated level set is drawn on top of the mesh.

Figure 8 shows three intermediate stages in the Stage 2 morph of the symbol "T".

In unison with one or both stages of the morph, each

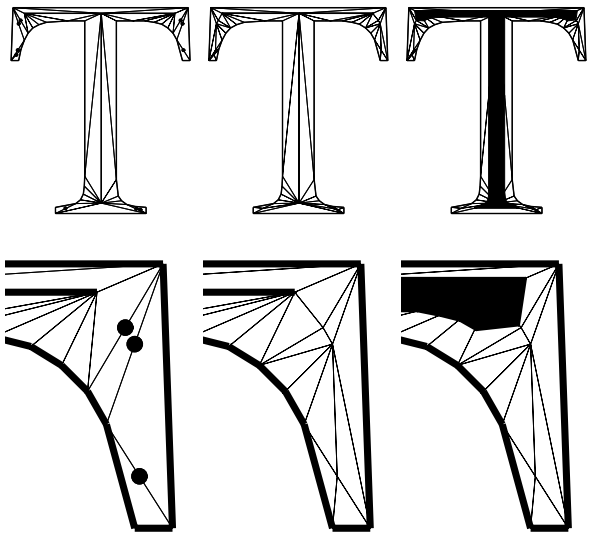

Figure 7: *Creating a Stage-2 morph for the letter "T" (top row) and a close-up in a serif region (bottom row). The triangulation at left respects the constraint edges indicated by thick lines. The points indicated in the left panel are added to the mesh shown at center. At right, an interpolated level set is shown.*

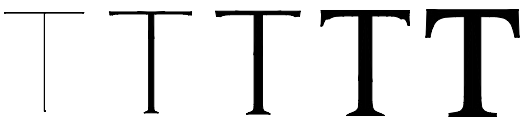

Figure 8: *A few steps in a Stage-2 morph.*

symbol is also gradually translated toward its final position, which is determined largely by neighboring symbols. That is, the relative positioning of the hand-drawn symbols is first used to determine the relation between them, such as right-neighbor, left-neighbor, subscript, and superscript. This layout information, along with spacing information associated with the outline font, is used to determine final positioning of the text.

## Conclusion

We have described a new technique for managing the appearance of hand-printed text in a pen-based system. The technique combines character recognition and shape metamorphosis (morphing) to create "smart text" that gradually reshapes itself into compact and legible type. The result is a qualitatively different environment for handwritten symbols that has advantages over traditional pen-based systems. In particular, sudden changes in appearance and organization are eliminated, and new techniques for error correction are suggested, as we describe in the future work section. The rate of morphing can also be adjusted to suit specific tasks or user preferences.

The goal of this work is to enhance pen-based user interfaces by providing recognition feedback and high legibility without distracting display artifacts. The primary benefit of gradual transformation is that the system appears to *enhance* one's writing rather than *replace* it. Moreover, since there are no abrupt changes, it is easier to maintain a mental map of the text despite alterations to shape, scaling, and spacing.

## Future Work

Many improvements and extensions can be made to our current system. Currently, our system applies the two stages of the morph in sequence. Ideally, these operations would be applied in parallel, which would either eliminate the intermediate stroke font, or blend the stage-1 and stage-2 morphs together so that both are applied at all intermediate stages of a morph sequence.

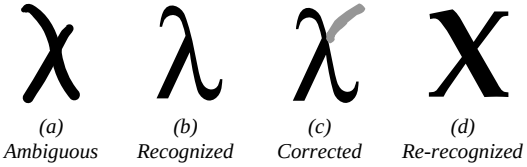


Figure 9: *A user-drawn symbol that is mis-recognized as a "λ." By touching up the raster image (shown in gray), the symbol is re-recognized as an "X".*

Better methods are needed to handle stroke matching; while the heuristics that we have described in this paper work well in the majority of cases, there are occasional instances where it makes a poor choice. The result is an unnatural-looking morph, and sometimes unintelligible intermediate forms. A more robust approach would be to consult the recognizer on one or more of the intermediate forms to ensure that they are legible, and to construct a better correspondence if they are not.

There are a number of error correction methods that would be very natural in our proposed system, such as the "touch up" method depicted in Figure 9. Here, the recognizer must handle slight alterations of existing outline fonts; both additions, as shown, and deletions indicated either by erasing or by scribbling out. The recognition task can still be performed by a simple feature-based approach, as we have employed in our system, by applying the modifications to the underlying stroke font and resubmitting it to the recognizer.

Finally, there are many unanswered questions about the use of smart text. At what rate should a morph occur? Should shape morphing be controlled independently from scaling and repositioning? How do these choices depend on context in which the text is written, or the preferences of the user? Clearly, extensive user studies will be need to discern the most appropriate uses of smart text, and the most effective parameter settings.

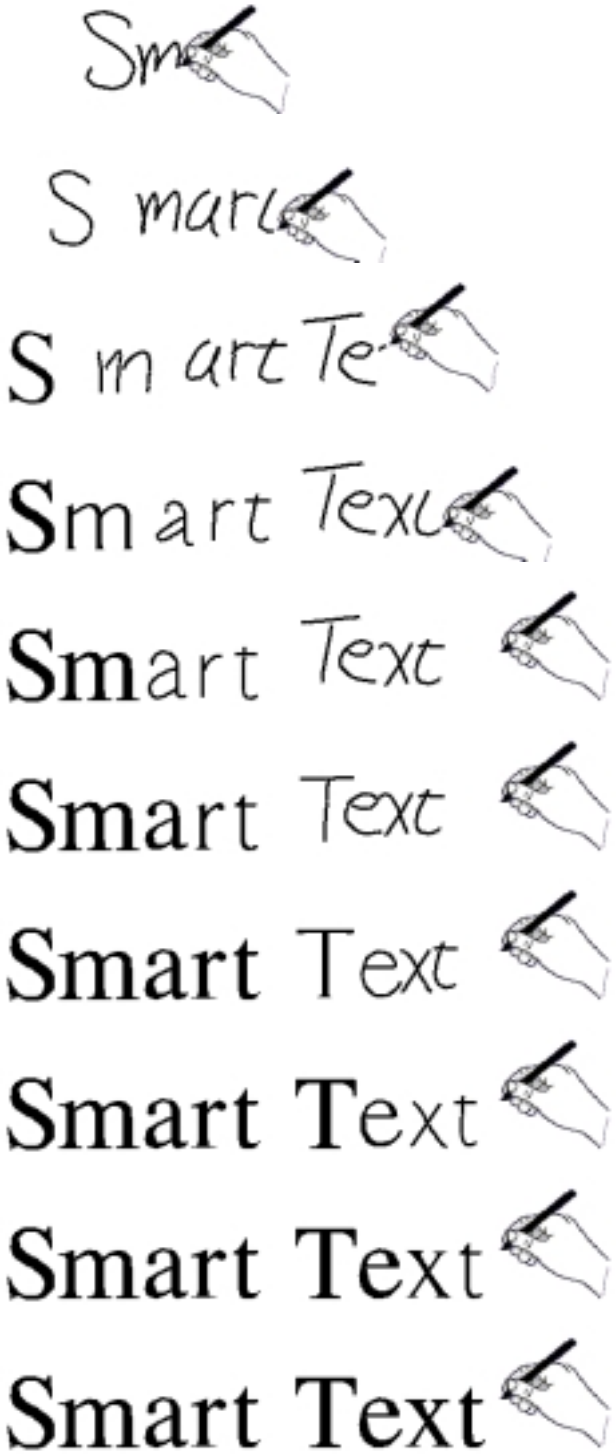


Figure 10: *A complete example in which hand-written text is gradually repositioned as it morphs into clean type. Note that the style of the "a" changes as it morphs, yet it remains legible.*

## References


Arvo, J. 1999. Computer aided serendipity: The role of autonomous assistants in problem solving. In *Proceedings of Graphics Interface '99*, 183–192.

Avitzur, R. 1992. Your own handprinting recognition engine. *Dr. Dobb's Journal* 32–37.

Beier, T., and Neely, S. 1992. Feature-based image metamorphosis. *Computer Graphics* 26(2):35–42.

Del Bimbo, A., and Pala, P. 1997. Visual image retrieval by elastic matching of user sketches. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19(2):121–132.

Lee, S.-Y.; Chwa, K.-Y.; Shin, S. Y.; and Wolberg, G. 1995. Image metamorphosis using snakes and free-form deformations. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 439–448.

Lorensen, W. E., and Cline, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics* 21(4):163–169.

MacKenzie, I. S., and Zhang, S. X. 1997. The immediate usability of graffiti. In *Proceedings of Graphics Interface '97*, 129–137.

Miller, E. G., and Viola, P. A. 1998. Ambiguity and constraint in mathematical expression recognition. In *Proceedings of the 15'th National Conference on Artificial Intelligence (AAAI-98)*, 784–791.

Novins, K., and Arvo, J. 1999. The morphological cross-dissolve. In *SIGGRAPH '99 Conference Abstracts and Applications*, 257.

Prasad, L. 1997. Morphological analysis of shapes. *CNLS Newsletter* 139:1–18. Also available as: http://cnls-www.lanl.gov/Highlights/1997-07.

Sederberg, T. W., and Greenwood, E. 1992. A physically based approach to 2-d shape blending. *Computer Graphics* 26(2):25–34.

Sederberg, T. W., and Greenwood, E. 1995. Shape blending of 2-D peicewise curves. In Dæhlen, M.; Lynche, T.; and Schumaker, L., eds., *Mathematical Methods for Curves and Surfaces*. Nashville: Vanderbilt University Press. 497–506.

Seitz, S. M., and Dyer, C. R. 1996. View morphing. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 21–30.

Smithies, S.; Novins, K.; and Arvo, J. 1999. A handwriting-based equation editor. In *Proceedings of Graphics Interface '99*, 84–91.

Turk, G., and O'Brien, J. F. 1999. Shape transformation using variational implicit functions. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 335–342.

Wolberg, G. 1982. *Digital Image Warping*. Los Alamitos, California: IEEE Computer Society Press.

Yaeger, L. S.; Webb, B. J.; and Lyon, R. F. 1998. Combining neural networks and context-driven search for online, printes handwriting recognition in the Newton. *AI Magazine* 19(1):73–89.

Zeleznik, R. C.; Herndon, K. P.; and Hughes, J. F. 1996. SKETCH: An interface for gestural modeling. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 163–170.