Stroke-by-Stroke Glyph Animation

Yotam Gingold
New York University
719 Broadway, New York, NY, USA
gingold@cs.nyu.edu

David Salesin
Adobe Systems and Univ. of Washington
Seattle, WA, USA
salesin@adobe.com

Denis Zorin
New York University
719 Broadway, New York, NY, USA
dzorin@cs.nyu.edu

Abstract

We present a technique for automatic generation of animations of font glyphs for video text effects and kinetic typography. Each glyph is animated as a sequence of strokes, imitating drawing the glyph with a pen. Starting with an annotated stroke skeleton for each letter, our algorithm automatically computes animations for corresponding glyphs for a broad variety of fonts. Our algorithm imposes few constraints on font style, and can handle fonts with complex and unusual outline shape and topology. The technique also has applications to embroidery and mosaics.

1. Introduction

Text animation is supported by many presentation and video editing tools. Examples of common effects include text motion, gradual appearance or dissolve of text, text deformation and many others. One particularly natural way to animate text is to make it appear as if it is written by hand. This type of font effect is particularly natural for fonts imitating handwriting, but can be used for practically any font, as most font glyphs can be reasonably well approximated by strokes. However, decomposing glyphs of a font into strokes manually requires considerable effort and skill. Quite often finding a good decomposition is difficult: while all fonts consist of glyphs easily recognized as letters of an alphabet, the variety of geometric shapes these characters can take may be large, with different topology, placement of sharp corners and decorative features (see Figure 1).

In this paper, we present an automatic technique for stroke-based glyph animation. For fonts representing relatively short alphabets (Latin, Greek, Hebrew, Cyrillic, Armenian, Georgian, Korean, to name some examples) one can create a library of skeletal glyph animations and then transfer these animations to other fonts automatically. Our goal is to be able to transfer font animation for a maximally broad variety of fonts styles.

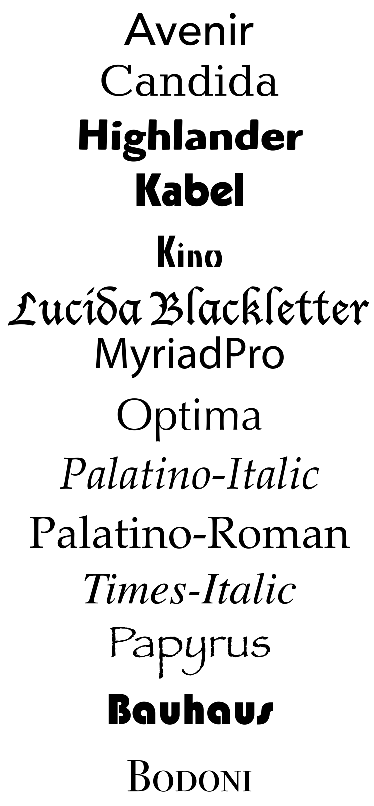

Figure 1. Example fonts.

Our algorithm requires little user intervention, other than specifying several per-font parameters. Any change made to the original animation sequence is easily propagated to any other font. In our work we have focused on the standard Latin alphabet, but the approach is applicable to most alphabets in existence (although its usefulness is limited for Chinese characters, due to a large number of glyphs for which the strokes have to be specified manually).

Conceptually, our technique can be divided into three phases: skeleton fitting, glyph segmentation, and animation. For fine-tuning animations—and to correct the glitches that sometimes occur—users can modify (or replace) the output of the skeleton fitting and glyph segmentation phases.

Finally, our algorithm has applications beyond glyph animation: both embroidery and mosaics containing text require a stroke-based decomposition of glyphs.

Previous work. There has been relatively little work on automatic glyph animation. In practice, it appears that glyph animation is usually done by hand, or automatically in a straightforward way using stroke-based character definition. However, most fonts are stored as outlines, with no explicit stroke information. Moreover, letters of most alphabet-based writing systems are not traditionally defined as a combination of strokes, as it is the case for Chinese characters.

Extracting skeletal strokes from fonts is more common ([11], [17], [18]), and is typically done with the purpose of recognizing hand-written or scanned characters, so the visual quality of strokes is not significant. In the case of skeletons, we solve a complimentary problem: the letter to which the glyph corresponds is known, and the goal is to match a standard skeleton to a particular glyph shape. [8] extracts important font elements (e.g. serifs) with the goal of generating a parametric representation of a glyph which can be used for hinting.

As glyphs can be viewed as images, techniques extracting strokes from painted images or approximating photographs with strokes can potentially be used [13, 3, 9, 15]. While these techniques can be applied to glyphs, in most cases they are unlikely to work well, as only relatively small number of precisely fitted strokes can imitate hand writing behavior, and such techniques are likely to generate a large number of fine strokes if precise reproduction of characters is needed.

Other related work on font stroke generation includes stylized stroke synthesis [14, 16]; these techniques assume that stroke skeletons are already defined, and focus on generating a particular stroke style. The work on computer-aided font design [5, 7] provides insight into typical techniques used to construct fonts starting from basic skeletons.

Finally, the work on animation transfer in character animation (e.g. [2, 12, 1]) solves a similar overall problem: a skeleton and its motion are adjusted to fit a different character geometry. However, these papers do not consider stroke decomposition which, in our case, is the central goal.

2. Overview

To motivate our algorithm, we consider typical variabilities in fonts that we would like to be able to handle.

We refer to the manually animated font as the *source font* and the font to which the animation is transferred as the *target font*. The target and source font may differ in any of these categories:

- weight, stretch and slant;
- filled-in or outline, as well as more unusual topology changes (broken outlines, segmented letters and other);
- serif and sans serif, more generally by decoration type.
- letter variant (e.g. the letter *a* has two dominant basic shapes).

This list is by no means exhaustive. While there is little hope that purely automatic transfer would work well for all exotic fonts, the goal of the algorithm design is to ensure that the algorithm makes as few assumptions as possible about the shape of target characters. In particular, we do not assume that the outline topology of the source and the target coincides, or that characters are composed out of strokes. We only make a weaker assumption that characters can be closely approximated by a sequence of strokes, possibly of complex shape.

The source animated glyphs are represented by their *skeleton graphs*, with strokes corresponding to paths through the graph. The four main phases of our algorithm are (1) skeleton fitting, (2) creation of *rough strokes*, which are thick and cover the whole glyph; (3) for each rough stroke, computation of a mask determining the actual stroke shape; (4) animation of refined strokes.

The goal of the first phase is to align the skeleton with the target character. The second phase inflates each skeletal stroke so that the glyph is covered and overlap between strokes is restricted to the skeletal stroke intersection areas; At the third stage, the mask is obtained by cutting out a part of the initial glyph inside the rough stroke which has maximally “stroke-like” shape. At the final step, rough strokes are animated using an association of points on the rough stroke outline with skeletal stroke points, with masks applied to display only points covered by the original glyph.

In the next sections we describe the sequential stages of the process in detail.

3. Skeleton fitting

The goal of this stage is to find the set of directed curves, possibly connected at the endpoints, representing the pen strokes for drawing the glyph. The *skeleton graph* S consists of curves s_i possibly sharing endpoints. In our implementation s_i are either line segments or chains of cubic Bezier segments parameterized on $[0,1]$ joined with C^2 continuity. Any other parametric curve representation can be used, however cubic Bezier segments have C^2 continuity, which appears necessary to approximate the variety of strokes in e.g. the Latin alphabet.

Each curve in the graph is a part of one or more directed paths through the graph, which we call *skeletal strokes*. Note that skeletal strokes can share parts. (The lower half of the 'h's stem in Figure 6 is shared by a downward skeletal stroke and an upward one.) The path can be annotated with the constraint that sequential curves share a tangent.

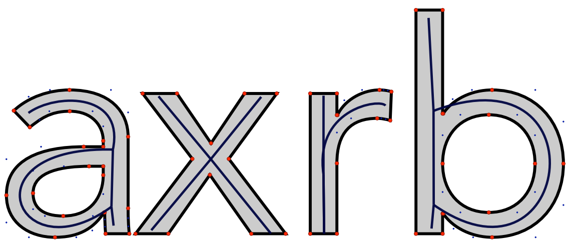

Figure 2. Skeletons of several letters. Curves and nodes can be shared by different strokes.

Given a target glyph to animate, we determine its stroke graph by fitting a template stroke graph for the letter, which we assume to be defined by hand. We treat this as an optimization problem, minimizing an energy measuring distances of points inside the target glyph to the skeleton, while maintaining tangential constraints between curves in strokes and minimizing skeleton stretching.

As a measure of the quality of fit, we use the average of a fast-decaying function of distances from the points inside the glyph to the skeleton S, specifically

$$E_{fit} = \int_{(x,y)\in glyph} -e^{-kd(x,y,S)^2}dxdy$$

where k is a constant chosen based on weight of the font (an heuristic procedure for choosing k is described below) and $d(x,y,S)$ is the distance from the point (x,y) to the skeleton S. The intuition behind this choice can be understood as follows. The expression $e^{-kd(x,y,S)^2}$ decays as a gaussian as the distance from S increases; the contours of this function (as d increases) are blurrier and blurrier version of S. That is, the expression $e^{-kd(x,y,S)^2}$ can be interpreted as a "sweep" of a 1D Gaussian along, and perpendicular to, the skeleton. Consider a line perpendicular to S at a smooth point $q \in S$. Then for any point (x,y) of this line closer to q than any other point of S (i.e. $d(x,y,S) = \|(x,y) - q\|$), in other words, along this line up to the medial axis, we get the usual one-dimensional Gaussian $\exp(-k\|(x,y)-q\|^2)$. Note that this Gaussian sweep is similar to well-known convolution surfaces (Figure 3), but does not suffer from the the undesirable thickening effect near curve joints visible near the crossing of 't'.

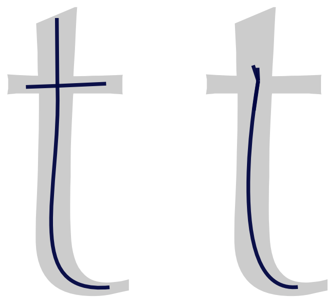

Figure 3. Gaussian sweep (left) and Gaussian convolution (right).

Our energy measures the overlap between the Gaussian sweep of the skeleton and the glyph by integrating the part of the sweep in the interior of the glyph. As the magnitude of the sweep is the largest near the skeleton, the energy minimization pushes the skeleton towards the interior.

Rather than using the standard Euclidean distance which is not differentiable at the points of the medial axis, we use an approximation: a *smoothed* distance function, which we define shortly, following [6]. This is a middle ground between convolution surfaces and pure Euclidian sweep that we just described.

The total energy contains two additional terms:

$$E = E_{fit} + w_{length} \sum_{s_i \in S, \hat{s}_i \in S^0} \int_0^1 \frac{\sqrt{1+s_i'(t)^2}}{\sqrt{1+\hat{s}_i'(t)^2}}dt + w_{tangent} \sum_{(s_i,s_j)\in S^t} arccos\left(\frac{s_i'(0)}{\|s_i'(0)\|}\cdot\frac{s_j'(1)}{\|s_j'(1)\|}\right)^2$$

where $s_i \in S$ are curves of the skeleton parameterized by $t \in [0,1]$, $\hat{s}_i \in S^0$ are the initial positions of the curves, S^t is the set of pairs of curves with are constrained to have common tangent and $w_{length}, w_{tangent}$ are weights. The second term ensures that the skeleton does not stretch.

(Minimization of E_{fit} without constraints would lead to a space-filling curve covering the glyph.) The last term is a penalty term with large weight enforcing the tangent constraint.

The constant k in E_{fit} is chosen so that $e^{-kd(x,y,S)^2}$ falls off to approximately .75 when d is half the average thickness of the glyph. (This varies with the weight of the font.) Too small a k encourages multiple curves of the skeleton to cover the same glyph strokes, while k too large sensitizes curves to distant regions of the glyph. When k is properly chosen, E_{fit} has little to gain as the skeleton double-covers the parts of the glyph. The constant w_{length} is set experimentally, to penalize such over-stretching. Note that thicker fonts have a larger $|E_{fit}|$, and w_{length} needs to be increased accordingly.

If we use minimal distance to S in the expression for E_{fit}, the energy is not a smooth function of the positions of the control points of the curves, which makes it impossible to apply gradient or Hessian-based optimization techniques. Instead we approximate the distance $d(x, y, S) = \min_{q \in S} \|(x, y) - q\|$ with a smoothed distance

$$d_p(x, y, S) = \left(\sum_{s \in S} \int_0^1 \|s(t) - (x, y)\|^{-p} dt \right)^{-1/p} \quad (1)$$

(See [6] for a detailed explanation of this approximation.) We found $p = 5$ to be adequate for our application.

To minimize E numerically, we discretize g on a grid whose long side is 64 and with similar aspect ratio to the glyph, and discretize the integral in d_p by sampling the curves of S. We compute d_p discretely, for each line segment of every stroke.

The energy we optimize is non-linear and relies on a reasonable initial placement of the stroke. To obtain a good initial position, we translate and scale the bounding box of the known glyph matching the stroke graph to match the bounding box of the target glyph. The same transformation is applied to the stroke graph to obtain its initial position.

Initial guesses and optimization results can be seen in Figure 4.

Defining skeleton graphs. While there are no formal restrictions on the user-defined skeleton graphs, several rules need to be observed to obtain good results. First, the number of degrees of freedom (control points of Bezier segments and end points of line segments) should be minimal.

Intersections of skeletal strokes must be modeled explicitly, i.e. a node must be inserted at the intersection. (Once skeleton fitting is complete, however, skeletal strokes with tangent constraints are fused together into a single skeletal stroke for the remainder of the algorithm.) A number of letters may have different letterforms (for example, *a* and *g* and even the italic *f*, whose stem hangs far below the baseline).

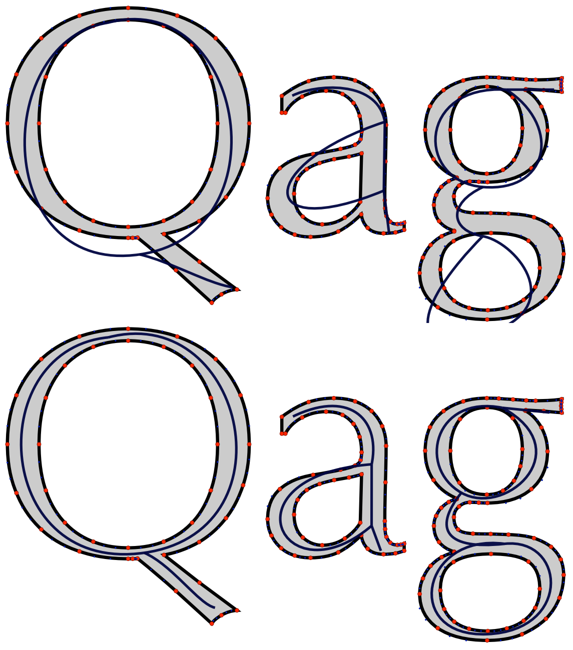

Figure 4. Initial guesses (top) and optimized placement (bottom) of skeletons.

4. Rough strokes

Rough strokes are an initial approximation to the decomposition of the character into thick strokes. We want them to have the following properties:

- Each rough stroke follows a skeletal stroke;
- All points of a glyph are covered by one or more rough strokes.
- Rough strokes intersect only near intersections of corresponding skeletal strokes.

These goals are achieved as follows. We *inflate* all strokes until the target glyph is covered by the union of all inflated strokes. The inflation is achieved by offsetting points along each stroke using the distance function to the stroke. The inflation is stopped as soon as the glyph is entirely covered, which is tested by rasterizing the inflated strokes and the glyph on a 100x100 grid and using pixelwise operations for the test (Figure 6).

The simplest way to inflate the strokes is to offset the stroke along the normal direction. However, such offsetting may result in self-intersections whenever a stroke has

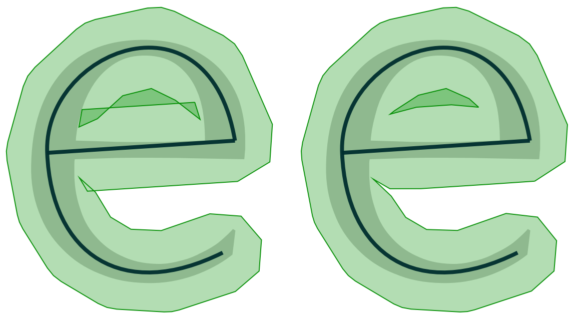

Figure 5. Offsetting along normals (left) and integral lines of the smoothed distance gradient field (right).

concave features (Figure 5); avoiding self-intersections requires computing the medial axis and limiting stroke inflation so that the medial axis is never intersected, which is relatively complicated. Fortunately, the smoothed distance function (Equation 1) helps to solve this problem in a simple way. We compute the offset by tracing the integral lines of the gradient field starting from each stroke. These integral lines never intersect, unless they reach a point where the gradient of the distance function is zero, which can be easily detected. (To avoid the discretization artifact mentioned in [6] of integral lines escaping between points of a discretized stroke, we compute the distance to line segments of the discretized stroke instead.)

From convex corners we trace three equally spaced points, from concave corners one, and from the start- and end-points of the stroke we trace five.

We note that the parts of the offset outline may have areas of high compression corresponding to the sharp concave corners on the skeleton. Once the offset outline is computed, we post-process the outline, removing sharp corners.

Note that the (1D) parameterization of a skeletal stroke naturally extends to the rough stroke inflated from it (Figure 7). Ignoring the start- and end-caps (created by tracing outward from the start- and end-points of the skeletal stroke), each point $s(t)$ on the skeletal stroke is identified with two points on the boundary of the rough stroke, the points traced from $s(t)$ in the positive and negative normal directions which we denote $trace(s(t)^{+})$ and $trace(s(t)^{-})$. The area associated with $s(t)$ then is the area enclosed by the strips of rough stroke boundary from $trace(s(0)^{+})$ to $trace(s(t)^{+})$ on one side and $trace(s(0)^{-})$ to $trace(s(t)^{-})$ on the other, connecting $trace(s(0)^{+})$ to $trace(s(0)^{-})$ and $trace(s(t)^{-})$ to $trace(s(t)^{+})$ with straight line segments.

To complete the parameterization of the rough stroke, we prepend and append the start- and end-caps, which we

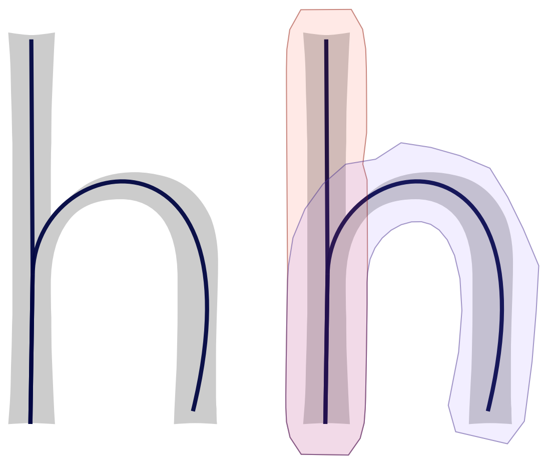

Figure 6. Rough stroke computation: the skeleton graph (left) and the inflated strokes (right).

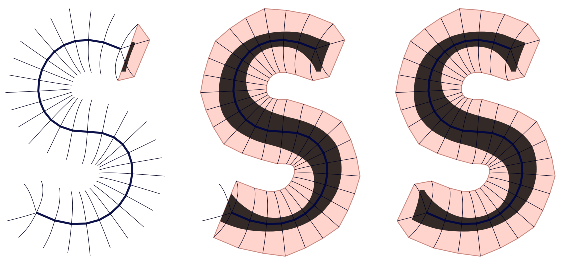

Figure 7. Parameterizing a single stroke.

also parameterize as follows. The start-cap is created by tracing two additional paths equally spaced between $s(0)^{+}$ and $s(0)^{-}$ (Figure 7, left); its parameterization begins at the midpoint between the line segment joining the ends of the two additional traced paths and expands along the cap boundary (in both directions) towards $trace(s(0)^{+})$ and $trace(s(0)^{-})$. The end-cap is created similarly, attaching to $trace(s(1)^{+})$ and $trace(s(1)^{-})$, and parameterized similarly, except in reverse. This composite parameterization of (start-cap, skeletal stroke, end-cap) devotes time proportional to the arc length of each, where the arc lengths of the start- and end-caps are the straight line distances from $s(0)$ and $s(1)$ to the beginning/end of the start-/end-caps' individual parameterizations.

5. Stroke masks

Once rough strokes are constructed, one can obtain an animation of the glyph.

Simply draw the inflated strokes in order, parameterizing each (inflated) rough stroke with the natural parameterization. If the original glyph is used as a mask for drawing strokes, we obtain refined strokes following glyph outlines. However, artifacts will appear at junctions wherever the outline has concave corners (Figure 8), typically, when two strokes intersect.

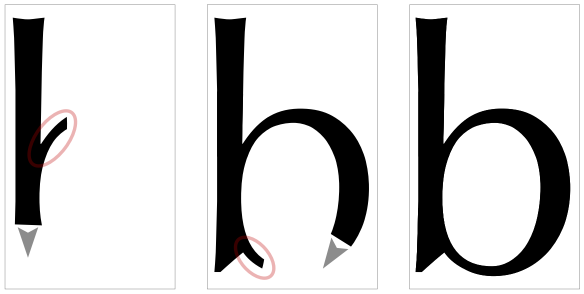

Figure 8. Three frames of an animation using rough stroke masks only. The grey arrow indicates the direction of animation. Artifacts at concave corners are circled in red.

The goal of this stage of the process is to reduce this type of artifact. We create an individual mask for each rough stroke, which we obtain by starting from the original glyph and then removing pieces. The refined strokes we use for our final animation are the rough strokes together with their individual masks.

The central observation is that a typical stroke has only convex corners (concave smooth features may be present). Concave corners typically appear where two strokes meet, although for some types of fonts may appear at any location. Sharp corners are easy to identify in glyphs: both Type 1 and TrueType fonts store outlines as oriented Bezier curves or straight line segments. Corners must be endpoints and are identified as concave by their tangents.

With appropriate cuts, we can eliminate concave corners. We use a greedy procedure to achieve this. At each concave corner point, we identify possible cuts between the corner and other points on the contour (not necessarily corners). We consider cuts between the corner and the closest *neighbor point* (if there is one), where a neighbor point is defined similarly to [10].

Two points are considered *stroke neighbors* if:

- There is no contour edge intersecting a straight line between them;
- The vector between the two points is close to one of the tangent directions at either point (we use $\pi/10$ as the threshold);

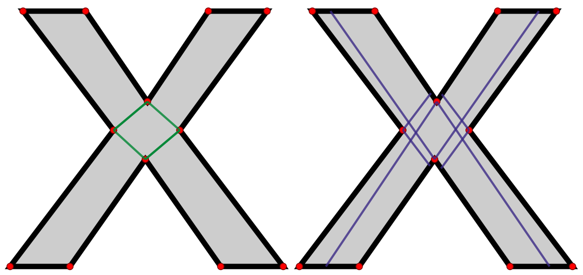

Figure 9. Examples of stroke neighbor cuts (left) and tangent direction cuts (right).

- One can select one of the two tangent vectors at each point so that the angle between them does not exceed a threshold (we use $\pi/5$).

To identify neighbor points we consider all points in a dense sampling of the glyph outline.

Note that neighbor points are often also concave corners. We also consider cuts in two tangent directions at the corner if they differ sufficiently from the direction towards the closest neighbor point. An example of cut letters can be seen in Figure 9.

In a greedy fashion, we make each cut, which necessarily creates two contours from one [1], and choose the side which contributes more energy to the stroke. Specifically, we clip each candidate letter against the rough stroke and then calculate E_{fit} energy between each candidate letter and the stroke. When calculating E_{fit} here, we use a very small k (5% of the value used for skeleton optimization). This is to further desensitize strokes to distant regions of the glyph, an issue which can occur on fonts with thin serifs (if serifs are modeled in the skeleton).

Note that there is no explicit guarantee that the union of all per-stroke masks covers the entire glyph. This can occur dramatically if different strokes choose the same side at a cut, although this does not occur with reasonable skeletons. However, this often leads to small artifacts where two strokes meet in a glyph as font designers often place a small filet that is a concave addition to either stroke.

6. Animation

The rough strokes and individual masks constructed in the previous steps are all that is needed to generate a glyph animation with refined strokes. Each rough stroke is assumed to be parameterized on the interval (t_i, t_{i+1}), and

[1]A careful reader may notice that contours must be simple (no holes) for this to work. We can remove holes with a hairline connection to an outer boundary, but care must be taken to place this hairline cut outside of the inflated stroke boundary whenever possible.

the strokes are arranged in a sequence. For each frame of the animation we draw the glyph clipped by the union of all inflated strokes intersected with their assigned masks. Finally, to avoid artifacts with filets, the entire unclipped glyph is drawn after the glyph's stroke-by-stroke animation completes.

Results with t_i determined by fraction of total arc length can be seen in Figure 10 and in the accompanying video.

7. Results

We have applied our technique to a variety of fonts of different styles and weights (Figure 10). In some cases, the automatic skeleton fitting algorithm produces bad results, but such skeletons can be easily corrected by hand, as each skeleton has only a small number of control points. In other cases, the per-stroke masks choose suboptimal cuts; these can also be easily corrected by hand, as each mask is created using a small number of cuts.

Fonts shown in Figure 10 as well as in the accompanying video are listed in the following table along with their skeleton fitting constants:

Font	letters	k	w_{length}
Avenir Medium	b,j,r,t	$10/300^2$	1.55
Candida Roman	q	$10/300^2$	1.55
Highlander Bold	l,n,o,v,y	$10/850^2$	6.4
Kabel Black	d	$10/850^2$	7.12
Kino	c,g,k,u	$10/300^2$	1.55
Lucida Blackletter	p,s	$10/600^2$	3.11
MyriadPro Regular	m	$10/300^2$	1.55
Optima Regular	f,h,i,x	$10/300^2$	1.55
Palatino Italic	e,z	$10/300^2$	1.55
Palatino Roman	A	$10/300^2$	1.55
Times Italic	w	$10/300^2$	1.55

During fitting, TrueType fonts are normalized to the Type 1 standard (1000 units per "em space" instead of 2048).

The following table presents the success rate of skeleton fitting and the overall animation appearance, measured per letter. (k and w_{length} are those specified in the table above.) A skeleton fit or animation is counted as successful if it lacks visual errors. (The animations shown in Figure 10 and in the accompanying video are all successful.) Note that an unsatisfactory skeleton fit nearly always leads to animation artifacts.

Font	animation	skeleton fitting
Avenir Medium	94%	100%
Candida Roman	92%	100%
Highlander Bold	86%	92%
Kabel Black	72%	81%
Kino	81%	92%
Lucida Blackletter	69%	89%
MyriadPro Regular	94%	100%
Optima Regular	92%	100%
Palatino Italic	92%	94%
Palatino Roman	86%	100%
Times Italic	86%	89%

Limitations. Some decorative fonts have outline behavior which does not match our assumptions about the role of concave corners. For example, the Bauhaus font outlines have no concave corners at all. In this case, points at which to originate cuts can't be determined by considering corners and the smooth outline itself must be analyzed with information from the stroke skeleton. The Papyrus font has outlines with many concave corners which do not correspond to strokes and whose tangents do not indicate useful cut directions or neighboring points. In this case, constructing precise strokes requires a much more complex analysis of glyphs.

8. Future Work

The robustness and speed of skeleton fitting can be improved with a multiresolution optimization process. Such a process would fit a skeleton with few degrees of freedom, and then iteratively introduce additional degrees of freedom and re-fit so long as doing so substantially enhances the fit.

It may be possible to improve the robustness of per-stroke masks using a snakes-based approach (also called active contours [4]). Such an approach would not be sensitive to outlines with too few concave corners (as in Bauhaus) or concave corners lacking meaningful tangent directions (Papyrus).

Modern fonts such as Bodoni are not well-handled by our system because of the radical thickness changes in strokes. To handle such fonts, the glyph could be non-uniformly weighted (perhaps using hinting information) to compensate for the difference in thickness (and thus area).

Finally, the skeleton fitting phase as well as the per-stroke mask computation are amenable to user input. An interface could be provided to allow the user to redraw some or all of the fitted stroke graph if the fitting phase produces an unsatisfactory fit. Similarly, if our algorithm is unable to find good cuts for a glyph's per-stroke masks, a user could sketch cuts onto the glyph to obtain better masks.

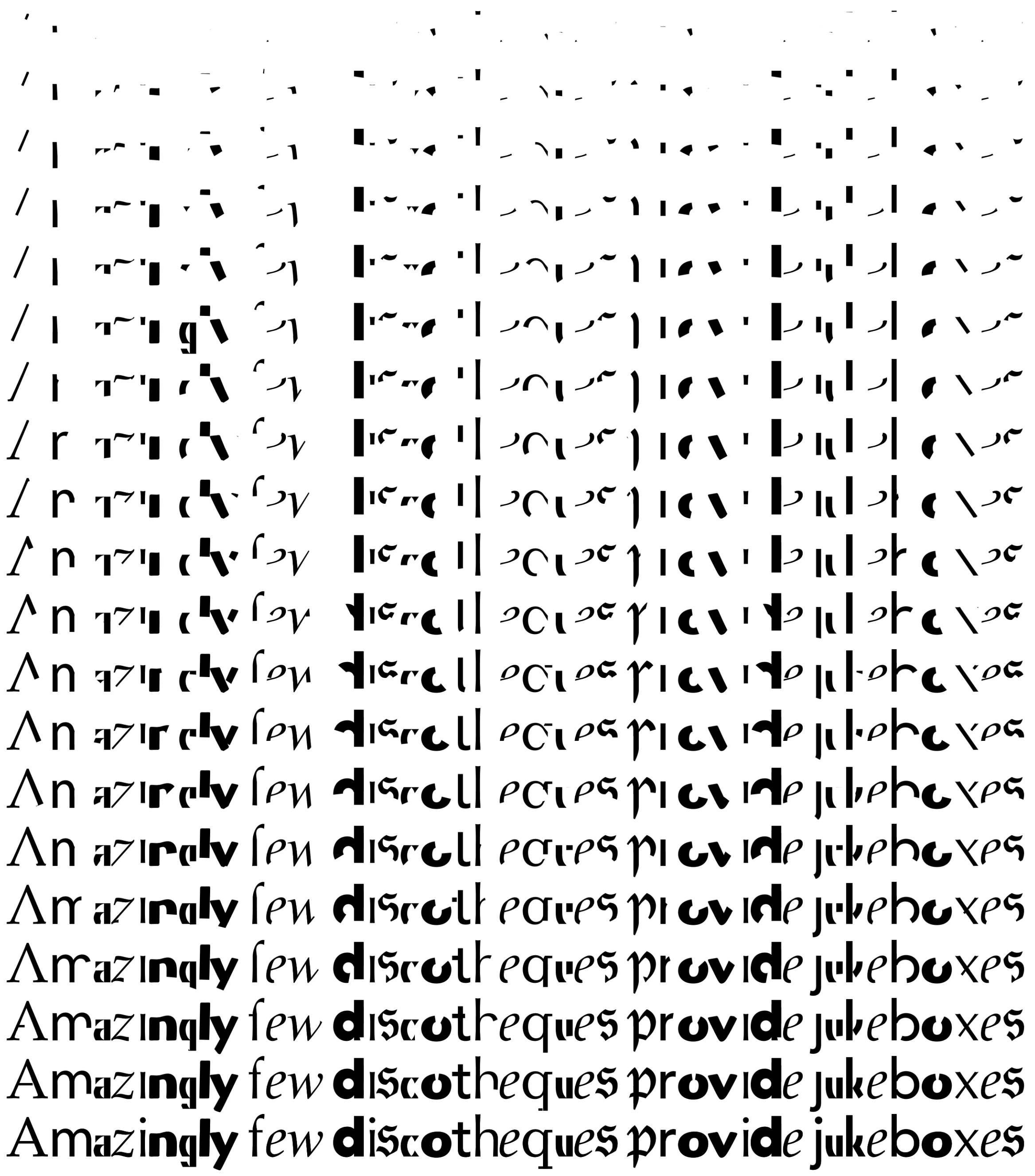

Figure 10. Examples of font animations.

References

[1] I. Baran and J. Popović. Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), August 2007.

[2] M. Gleicher. Retargetting motion to new characters. In *Proceedings of SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, pages 33–42, July 1998.

[3] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, pages 453–460, July 1998.

[4] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[5] Z. Pan, X. Ma, M. Zhang, and J. Shi. Chinese font composition method based on algebraic system of geometric shapes. *Computers & Graphics*, 21(3):321–328, May 1997.

[6] J. Peng, D. Kristjansson, and D. Zorin. Interactive modeling of topologically complex geometric detail. *ACM Transactions on Graphics*, 23(3):635–643, Aug. 2004.

[7] U. Schneider. A hybrid approach for stroke-based letterform composition including outline-based methods. *Computer Graphics Forum*, 19(4):243–256, Dec. 2000.

[8] A. Shamir and A. Rappoport. Extraction of typographic elements from outline representations of fonts. *Computer Graphics Forum*, 15(3):259–268, Aug. 1996.

[9] S. Simhon and G. Dudek. Pen stroke extraction and refinement using learned models. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 73–79, Aug. 2004.

[10] X. Song, Y. Luo, A. Niwa, and H. Ueno. Stroke Extraction as the Preprocessing Step for CJK Outline Font Compression. In *Proceedings of the 8th International Conference on Neural Information Processing*, 2001.

[11] Y. Su and J. Wang. A novel stroke extraction method for Chinese characters using Gabor filters. *Pattern Recognition*, 36(3):635–647, 2003.

[12] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, Aug. 2004.

[13] G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 1994*, Computer Graphics Proceedings, Annual Conference Series, pages 91–100, July 1994.

[14] H. T. Wong and H. H. Ip. Virtual brush: a model-based synthesis of chinese calligraphy. *Computers & Graphics*, 24(1):99–113, Feb. 2000.

[15] S. Xu, Y. Xu, S. B. Kang, D. H. Salesin, Y. Pan, and H.-Y. Shum. Animating chinese paintings through stroke-based decomposition. *ACM Transactions on Graphics*, 25(2):239–267, Apr. 2006.

[16] J. Yu and Q. Peng. Realistic synthesis of cao shu of chinese calligraphy. *Computers & Graphics*, 29(1):145–153, Feb. 2005.

[17] J. Zeng and Z. Liu. Stroke Segmentation of Chinese Characters Using Markov Random Fields. *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)-Volume 01*, pages 868–871, 2006.

[18] A. Zongker. Representation and recognition of handwritten digits using deformable templates. *IEEE Trans. PAMI*, 19(12):1386–1391, 1997.