Example-Based Automatic Font Generation

Rapee Suveeranont and Takeo Igarashi

Department of Computer Science, The University of Tokyo, Japan
iiieyes@gmail.com, takeo@acm.org

Abstract. It is difficult and time-consuming to create and maintain a consistent style in all characters in manual font design process. Thus, we present a method for automatically generating a new font from a user-defined example. From each character outline, a skeleton is derived to be used as a topological structure. The system can then represent an arbitrary font using a weighted blend of outlines and skeletons from template fonts. The system takes the outline of a single character drawn by the user and computes blending weights from a database to reproduce the given outline. The weights are then applied to all characters to synthesize the new font. In this paper, the algorithm is described in detail and its quality evaluated with some examples.

1 Introduction

The digital font design process begins with the sketching of all characters on paper. The designer then converts the characters into vector outlines and manually edits each character using a font-editing application. This is very time consuming and requires professional skill, artistic sense, hands-on experience, and expertise in typography, a combination of talents that few people other than professional font designers possess. Therefore, an automated process for generating fonts from a few examples is desirable.

Xu et al. [13] developed a system that automatically generates Chinese calligraphy. Their system is well suited to calligraphic languages thanks to its internal stroke-based font representation. Unfortunately, their method is not appropriate for characters from other language families, such as those that use a Latin alphabet.

Our research uses the work of Xu et al. as a basis but focuses on Latin alphabets and scalable outline fonts, which, compared to stroke-based fonts, lend themselves better to detailed design. Our goal is to create a system that facilitates the font design process by automating character creation with minimal user intervention. We take a single character outline as an input to generate a complete character set. The contributions of our research include a two-step font generation and a style preservation technique. Based on a skin and skeleton model, we introduce a morphable font model that is parameterized by blending weights. The style of the example character can be approximated by searching for the most similar blended style in a template database. Based on the premise that a few key characters hold most of the important features, our system enables the user to infer all other characters and generate them from the weights.

Because only one example is required, our system enables both amateurs and professional font designers to rapidly and effortlessly create new fonts. We demonstrate the effectiveness and capability of our method here.

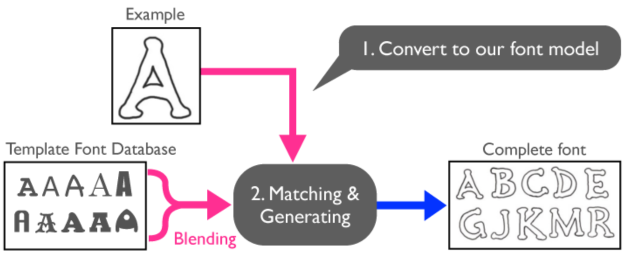


Fig. 1. Overview of our automatic font generation system

2 Related Work

Font Representation. Alternative font representations to outline-baed include feature-based and stroke-based. Shamir and Rappoport [1] proposed a feature-based approach that views fonts as high-level features such as serifs, bars, arcs and joints, rather than as low-level objects such as points and splines. This approach is strengthened by the component-based approach [2], which constructs fonts by using building block of parameterizable template shapes. Stroke-based representation of typographic character has been extensively studied because of its intuitiveness in characterizing calligraphic alphabets such as Chinese and Japanese. This approach provides complete topological information but is deficient in complex outline presentation. Many researchers have attempted to enhance the technique using concepts such as stylized stroke fonts [3] and skeletal strokes [4].

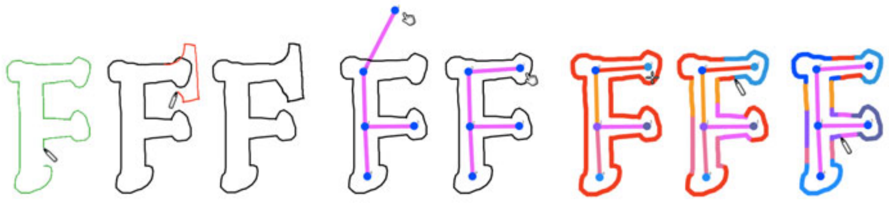
Shape-Preserving Deformation. One of our goals was to preserve the original style of the example character. Applications in 2D computer graphics usually involve object transformation or deformation. The original appearances of an object is usually depreciated by linear operations, which is undesirable. A number of studies has been undertaken to overcome this [4,5,6,9,10,11], many of which have tried to preserve a certain aspect of a 2D shape, such as area [5] or the original shape [6]. Sorkine et al. [10], used the Laplacian of a mesh to achieve rotation and scale-invariant surface editing. Nealen et al. [11] extended the idea and presented more general procedure using the Laplacian and positional constraints. Their approach motivated us to develop a Laplacian method for 2D outline that preserves the detailed features of a font (such as serifs, corners and junctions) and is suitable for text fonts.

Example-based Generation. Blanz and Vetter's [12] technique for creating a 3D human face from a 2D example facial picture is particularly noteworthy.

They presented the idea of a morphable face model that is searchable to match the input. Xu et al. [13] developed algorithms to automatically generate Chinese calligraphy. Their method works well for calligraphic scripts but cannot capture the complex designs of fonts used in printing and displaying.

3 Overview

We developed an easy-to-use graphical user interface that allows the user to draw an example outline (Figure 2). To edit the outline, the user can oversketch to replace existing outline. Our system automatically computes the skeleton and the correspondence between the skeleton and the outline. If the computed results are unsatisfactory, the user can modify them manually. When the user clicks a button, our system computes and generates a complete font.



(a) Draw and correct outline (b) Adjust skeleton (c) Select skeleton and paint correspondence on outline

Fig. 2. Our system's simple user interface

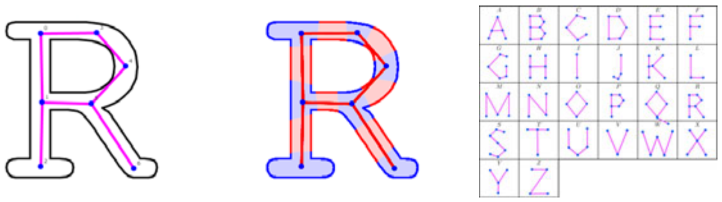
4 Morphable Font Model

4.1 Glyph Representation

The system uses a hybrid representation consisting of two components: a detailed profile skin and a structural skeleton. These two components permit the user to independently change the fine decorative style and the overall topological of the figure. For example, consider the K glyph with its original skin and skeleton. One can freely move, resize, or rotate the skeleton, and the skin is effectively deformed according to the skeleton but maintains its characteristic style. We call this *skin transfer*.

4.2 Correspondence of Skin and Skeleton

As a prerequisite for blending, the system requires a compatible skin correspondence for all template glyphs. An inappropriate correspondence produces badly blended outlines. Because there is no trivial universal skin correspondence-matching algorithm, a customized algorithm is required for each application. The



(a) Skin and skeleton (b) Correspondence (c) Skeleton model

Fig. 3. Glyph components. The skin and skeleton are rendered as black outline, pink edges, and blue vertices (a). Their correspondence: main features (blue) and internal features (red). (b) Note that the main features (rather than the internal features) are usually those that hold the decorative outline of the font. (c) Skeleton model.

system avoids conflicting topology matching between various font styles of the same alphabet by using one standard skeleton model.

In our initial observations of numerous fonts, we noticed that stylized portions of the skin usually enclose some skeleton vertices, which may be terminals, junctions or corner of the topology. Accordingly, outlines around skeleton vertices can be thought as more important (or *main features*), whereas outlines around skeleton edges, which are mostly straight lines and curves, can be thought of as less important(or *internal features*), as shown in Figure 3(b). Hence, the system matches the skin to either the vertices or the edges of the skeleton in accordance with its topological structure. Together with the skeleton model, we have a constructed compatible correspondence for all characters that conform to the model.

4.3 Morphable Model

We present a *morphable font model* that defines arbitrary fonts in terms of two types of blending weights: skeleton weights and skin weights. Let $F = \{f_1, f_2, \ldots, f_n\}$ be a set of template fonts f_i from a database. We define an arbitrary blending style S and a skeleton T using the following linear blending functions:

$$S = \mathrm{BlendStyle}_F(\Omega_S) = \sum_i^F \omega_{Si} \cdot \mathit{Skin}_{f_i} \tag{1}$$

$$T = \mathrm{BlendSkeleton}_F(\Omega_T) = \sum_i^F \omega_{Ti} \cdot \mathit{Skeleton}_{f_i} \tag{2}$$

$\Omega_S = \{\omega_{S1}, \omega_{S2}, \ldots \omega_{Sn}\}$, is a set of blending weights for skin, and $\Omega_T = \{\omega_{T1}, \omega_{T2}, \ldots \omega_{Tn}\}$ is for skeleton. For skin, blending is performed by first sampling corresponding skin outlines with the same number of vertices and then by linearly blending the vertex positions. For skeleton, this operation is performed only on skeleton vertex positions.

5 Algorithm

Font generation is accomplished in two steps: conversion of the example character to a morphable model and estimation of its blending weights.

5.1 Conversion to Morphable Model

The system first converts example character outline to our model, using a semi-automatic method. When automatic conversion yields unsatisfactory results, the user can manually modify them.

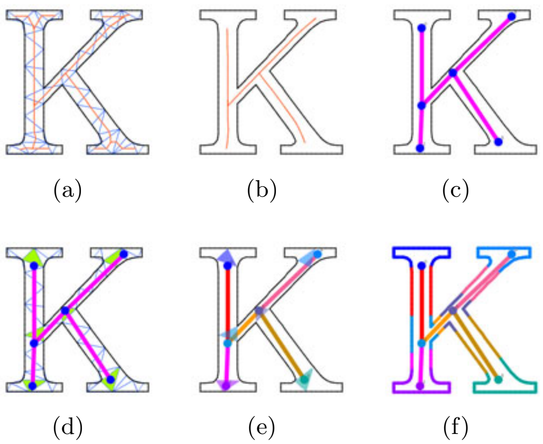


Fig. 4. Skeleton extraction and correspondences estimation. (a) Chordal axis. (b) Remove small branches. (c) Embed skeleton. (d) Nearest internal triangles. (e) Colorize skeleton features. (f) Fill correspondence.

Skeleton Extraction. We first extract a skeleton from a given outline based on a predefined skeleton model. The goal is to adjust the skeleton model so that it fits inside the outline and approximates the overall figure as accurately as possible. We compute the chordal axis transform of triangulation of the outline (Figure 4(a)) as introduced in [7,8], and then remove small branches and make it smooth (Figure 4(b)). We first exhaustively search for terminal and junction nodes first. Nodes of degree two can then be computed by following along the shortest path.

Skin–Skeleton Correspondence. Next, we establish a correspondence between the skin and the skeleton based on the triangulation of the previous step. The goal is to assign a certain portion of the skin outline to either a vertex or edge of the skeleton. Our motivation was the use of influence on attracted mesh vertices in the skeleton extraction technique proposed by Au et al. [14]. In our case, the mesh has the desirable property that its faces expand to full stroke breadth. Consequently, each triangle or edge is surrounded by skin points on every side so that they effectively group portions of the outline. For each skeleton vertex, we find the nearest internal triangle within a fixed range (Figure 4(d)). If there is no such triangle, an edge is used instead. We begin by establishing the corresponding skin at all points on such triangles or edges (Figure 4(e)). Finally, we flood-fill the remaining portion of the skin sequentially between each pair of established points (Figure 4(f)).

Feature-Preserving Skin Transfer. We blend the skin and skeleton separately (Section 5.2) and then combine the two to generate the final outline of a character. Applying simple linear transformation locally to the skeleton edges yields undesirable scaled and rotated effects, whereas fixing only the main features produces a broken outline (Figure 5(b)). We use a feature-preserving stitching technique to handle these problems [10]. We stitch the main and internal features and recover to the original style by performing a Laplacian-based editing operation on the skin (Figure 5(c)). The Laplacian coordinates are relative coordinates defined by the difference between a vertex and the average of its neighbors. Let $V = \{v_i\}$ be the set of original vertices on the skin. We can define the Laplacian coordinates of the vertex v_i using uniform weights (Figure 5(d)); specifically, $\mathcal{L}(v_i) = \frac{1}{2}(v_{i+1} + v_{i-1}) - v_i$. The feature-preserving stitching is then expressed by:

$$\begin{bmatrix} W_L L \\ W_P \end{bmatrix} V' = \begin{bmatrix} \mathcal{L} \\ W_P V \end{bmatrix} \tag{3}$$

Solving Equation 3 in a least-square sense yields a new skin that retains the original style. There are two types of constraints here: $W_L L V' = \mathcal{L}$ is the Laplacian constraint, and $W_P V' = W_P V$ is the positional constraint. W_L and W_P are diagonal weighting matrices. We set $W_{P_{ii}} = 1$, whereas $W_{L_{ii}}$ ranges from 1 to 10 depending on whether the vertex v_i is in the middle of a main feature ($W_{L_{ii}}$=1) or inside an internal feature ($W_{L_{ii}}$=10). These weights keep the main features fixed and the internal features smoothly connected.

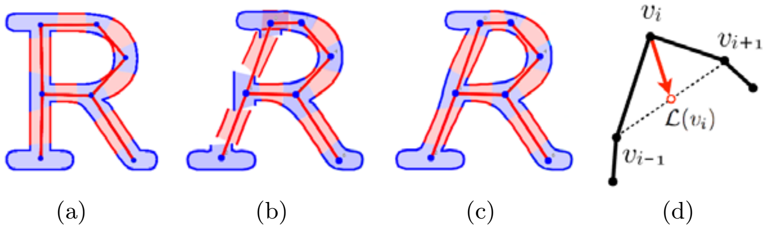


Fig. 5. (a) Effects of different transformations on the original glyph. (b) Simply fixing main features generates a broken outline. (c) Feature preservation using our method. (d) Laplacian coordinates.

5.2 Estimation of Blending Weights

Database. We prepare a database by converting all template fonts to our model using the method described in Section 5.1. If there are poor skeletons or correspondences, we fix them manually. When interpolating a font in the database, one can save memory by loading only the required characters from each font one at a time.

Common Coordinate. Before we can match a target style, we need to ensure that the matching is performed on the same font size. We use the cap height of the font (the distance from the base line to the top of a capital letter) as a common coordinate because it is consistent in every glyph, unlike width and line height.

Style Matching. In this step, the skin–skeleton representation of the example outline is taken as an input and the blending weights Ω_T for the structure are found, followed by the blending weights Ω_S for the style. Because it is impossible to find a globally optimal choice of weights (because of the exponentially large search space involved), we try to find an approximately optimal solution. We use the gradient descent method to find the solution that yields the minimum value of the penalty functions $I(\Omega_T)$ for the skeleton, and $J(\Omega_S)$ for the style, where $J(\Omega_S) = \alpha \cdot D(\Omega_S) + \beta \cdot A(\Omega_S)$, α, β are constants, $D(\Omega_S)$ is the total Euclidean distance, and $A(\Omega_S)$ is the total angular difference between corresponding points on the skin of the blended fonts and target font. The algorithm is simple: We start with a set of random weight parameters $\Omega_{S_1} \ldots \Omega_{S_M}$, based on the assumption that only some fonts contribute significantly to the optimal solution. Accordingly, we first determine the M template fonts that produce the smallest penalties. Each startup parameter occupies different portions of the blending space. We add these parameters to the queue. Picking the weights with smallest penaltyΩ' from the queue, we expand it to $N * 2$ new candidates, where N is the number of template fonts. The new weights are obtained by increasing or decreasing contributing weight from each template font by a small value ρ:

$$\Omega_i^* = \{\omega_1', .., \omega_i' \pm \rho\,, .., \omega_N'\} \tag{4}$$

$$Queue \leftarrow \arg\max_{\Omega_i^*} \big(J(\Omega_i^*) - J(\Omega')\big) \tag{5}$$

We add only weights for which the penalty value decreases by more than some threshold—that is, weights that seem to follow the fastest route to the solution. The algorithm repeats these steps until time runs out or the queue becomes empty. When the algorithm terminates, the resulting local optimal blending weights represent the style of the example. For the skeleton, $I(\Omega_T)$ is defined similarly.

6 Results

Our system was implemented in JAVA and all experiments were conducted on a computer with a 2GHz CPU and 1.5 GB of RAM. We set up the template font database beforehand, which consisted of 32 template fonts selected to represent various styles, such as serif, san serif, comic, and fancy, to examine the versatility of the system.

6.1 Automatic Font Generation

We tested our system by providing sample characters and observing the outputs. All of the example designs were created with our system. The fonts generated are shown in Figure 6.

Table 1. Time used for automatic font generation for each experiment

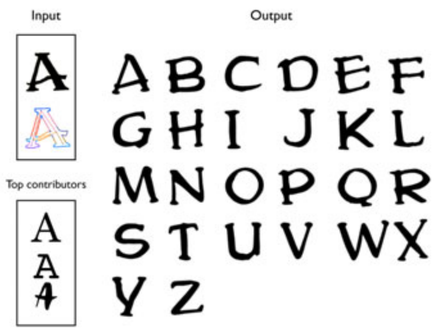
Experiment character	Results		
	Matching time (s)		Generating time (s)
	Skeleton	Style	
A	0.98	5.63	63.48
M	1.42	9.52	69.93
F	1.03	6.20	64.27
K	1.20	6.98	69.09

The generated glyphs in Figure 6(a) and 6(b) consistently appear to have the same style. The design of every characters represents the original glyph well. The overall dimensions—width, height, and slant—are consistent through out the fonts. This suggests that the skeleton estimation is reliable in most scenarios. Note also the consistent thickness of each font.

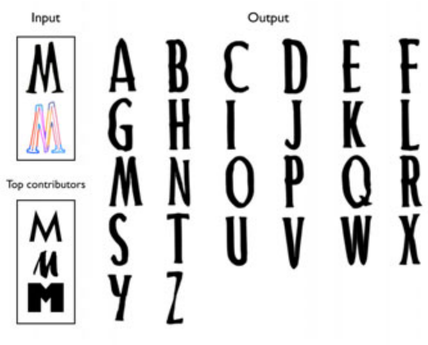
However, there is an obvious problem with linear blending in creating a novel design. The failed case in Figure 6(c) suggests that even when the top contributing fonts are promising (the blending of the first and second should be close to the example), the results can be far from perfect. If the template database is small, it cannot provide a precise estimate of the example style, and thus the system was unable to approximate the curvy corner of F in Figure 6(d). Paradoxically, this remains a problem even as the database gets larger. Specifically, contributions from most of the templates are rendered less significant because their results are averaged out by too many styles. One solution is to use only a few significant styles. Another might be to find different blending techniques that accumulate the contributions without deteriorative effects.

6.2 User Test

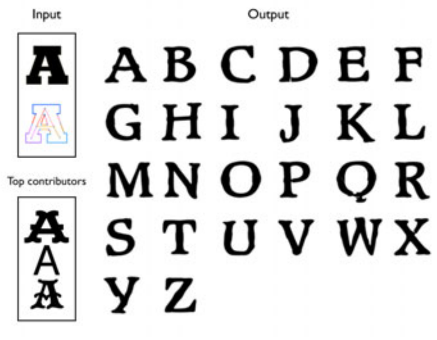
We conducted three informal experiments, each involving a single user. All of the users were interested in font design but had little or no practical experience. After a brief introduction, we asked each user to design three characters with different styles and to generate fonts from them. There was no time limit. After the test, the users were asked to evaluate our system on a scale of 1 to 5. We present the results in Figure 6(e), Figure 6(f), and Table 2.



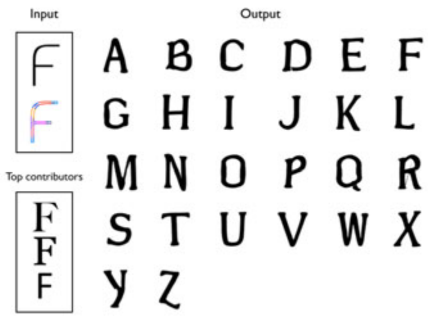
(a)



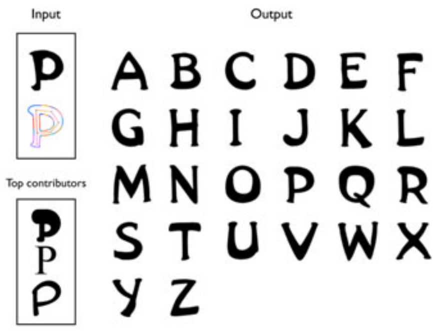
(b)



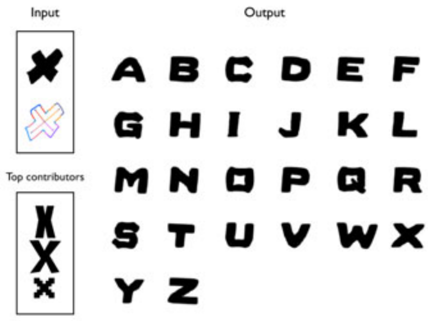
(c)



(d)



(e)



(f)

Fig. 6. For each experiment, the example outlines and correspondences (top) and the generated fonts (right) are shown. The three template fonts that contribute the most to blending weight of skin are listed in the bottom left box. (a)(b) Our generated fonts. (c)(d) Failed cases. (e)(f) Fonts generated by users.

The average time required to finish a drawing was less than 3 min, and even the longest experiment took less than 5 min. This suggests that our system is very fast compared to traditional process. One user commented that the system is very simple and straightforward because it requires only one outline. We also observe that after the users generated one font, they attempted to edit other characters without being given any instructions in this regard. This suggests that in an interactive font generation system, the user should be able to repeatedly revise his or her design for as many characters as he or she wants.

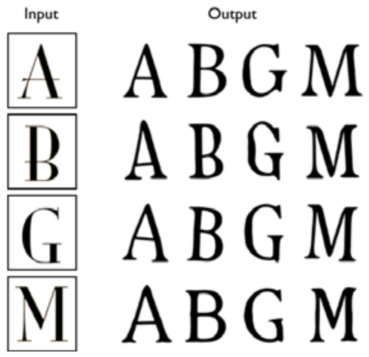
Table 2. Results of the user tests

User	Time to finish drawing (s)				Evaluation
	1	2	3	Average	
A	210	121	105	145.3	3
B	285	240	213	246.0	3
C	130	78	81	96.3	3
	Total average			162.5	3

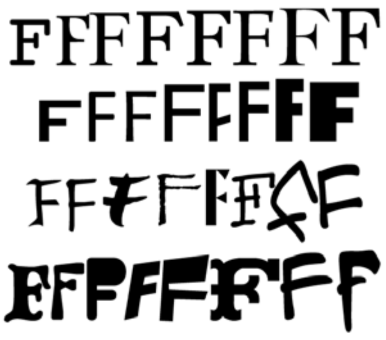
6.3 Consistency

We conducted a test to check the consistency of our method. To do this, we prepared a professional font that was not included in our template database. We then input each character into our system one at a time, and recorded the glyphs that were generated. If the generated glyphs from each input character had the same outlines, we could imply that our system produces consistent and reliable results.

The samples used in the tests are shown in Figure 7(a). In general, all test cases produced comparable styles. The structural aspects were also consistent despite small differences in the angles in each output.



(a)



(b)

Fig. 7. (a) Experiment on the consistency of multiple examples of the same style. Given the input examples (leftmost column), the output characters are generated (right columns). (b) Template fonts in our database.

7 Limitations and Future Work

Our current system is suitable for creating body text fonts, but not for fancy or stylized ones. Topological differences present one of the most difficult problems. The model should be more flexible and should permit correspondence matching even when such topological differences are present. In certain cases, such as small g and *g* (which have different topologies), it is debatable whether there should even be a correspondence.

Our example-based method lacks the ability to create a completely novel design. To generate such fonts with higher reliability, one must use a better blending technique that does not average out main-feature skin. Even though our system is capable of generating many styles, it cannot faithfully produce every style a user can conceive.

In addition to single example, we would like to extend the applicability of our system for multiple example inputs. We intend to redesign our morphable model and matching algorithm to reflect this concept. Such an approach would lead to a more accurate solution for each iteration. Our current implementation of the target style optimization step can be further improved in terms of speed and memory footprint. Using a better data structure or a faster algorithm would improve the overall performance. In addition, a practical optimization technique that moves toward a global optimum (such as stimulated annealing) should be tested. Finally, we wish to extend the database to include a greater variety of styles so that our system can generate fonts more accurately.

References

1. Shamir, A., Rappoport, A.: Feature-based design of fonts using constraints. In: Hersch, R.D., André, J., Brown, H. (eds.) RIDT 1998 and EPub 1998. LNCS, vol. 1375, pp. 93–108. Springer, Heidelberg (1998)
2. Hu, C., Hersch, R.D.: Parameterizable fonts based on shape components. IEEE Comput. Graph. Appl. 21(3), 70–85 (2001)
3. Jakubiak, E.J., Perry, R.N., Frisken, S.F.: An improved representation for stroke-based fonts. In: SIGGRAPH 2006: ACM SIGGRAPH 2006 Sketches, p. 137. ACM, New York (2006)
4. Hsu, S.C., Lee, I.H.H., Wiseman, N.E.: Skeletal strokes. In: UIST 1993: Proceedings of the 6th annual ACM symposium on User interface software and technology, pp. 197–206. ACM, New York (1993)
5. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 157–164. ACM Press/Addison-Wesley Publishing Co., New York (2000)
6. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shape manipulation. In: SIGGRAPH 2005: ACM SIGGRAPH 2005 Papers, pp. 1134–1141. ACM, New York (2005)
7. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: a sketching interface for 3d freeform design. In: SIGGRAPH 1999: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 409–416. ACM Press/Addison-Wesley Publishing Co., New York (1999)

8. Levet, F., Granier, X.: Improved skeleton extraction and surface generation for sketch-based modeling. In: GI 2007: Proceedings of Graphics Interface 2007, pp. 27–33. ACM, New York (2007)
9. Shamir, A., Rappoport, A.: Compacting oriental fonts by optimizing parametric elements. The Visual Computer 15(6), 302–318 (1999)
10. Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., Seidel, H.-P.: Laplacian surface editing. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, pp. 175–184. ACM, New York (2004)
11. Nealen, A., Igarashi, T., Sorkine, O., Alexa, M.: Laplacian mesh optimization. In: GRAPHITE 2006: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, pp. 381–389. ACM, New York (2006)
12. Blanz, V., Vetter, T.: A morphable model for the synthesis of 3d faces. In: SIGGRAPH 1999: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 187–194. ACM Press/Addison-Wesley Publishing Co., New York (1999)
13. Xu, S., Lau, F.C.M., Cheung, K.-W., Pan, Y.: Automatic generation of artistic Chinese calligraphy. In: IAAI 2004: Proceedings of the 16th conference on Innovative applications of artificial intelligence, pp. 937–942. AAAI Press/The MIT Press (2004)
14. Au, O.K.-C., Tai, C.-L., Chu, H.-K., Cohen-Or, D., Lee, T.-Y.: Skeleton extraction by mesh contraction. In: SIGGRAPH 2008: ACM SIGGRAPH 2008 papers, pp. 1–10. ACM, New York (2008)