“THEME ARTICLE”, “FEATURE ARTICLE”, or “COLUMN” goes here: The theme topic or column/department name goes after the colon.

Automatic Generation of Typographic Font from Small Font Subset

The automated generation of fonts containing a large number of characters is in high demand. For example, a typical Japanese font requires over 1,000 characters. Unfortunately, professional typographers create the majority of fonts, resulting in significant financial and time investments for font generation. The main contribution of this paper is the development of a method that automatically generates a target typographic font containing thousands of characters, from a small subset of character images in the target font. We generate characters other than the subset so that a complete font is obtained. We propose a novel font generation method with the capability to deal with various fonts, including a font composed of distinctive strokes, which are difficult for existing methods to handle. We demonstrated the proposed method by generating 2,965 characters in 47 fonts. Moreover, objective and subjective evaluations verified that the generated characters are similar to the original characters.

Tomo Miyazaki
Tatsunori Tsuchiya
Yoshihiro Sugaya
Shinichiro Omachi
Tohoku University

Masakazu Iwamura
Osaka Prefecture University

Seiichi Uchida
Kyushu University

Koichi Kise
Osaka Prefecture University

Typographic font creation is professional and time-consuming work. All characters in a font need to be consistent in terms of style and size; this is a task that is generally performed by humans. Automatic generation of fonts with large character sets is important for lowering design costs.

The problem of cost is particularly important in languages containing a very large number of characters, such as Chinese and Japanese. For example, most commercial fonts contain at least 2,965 kanji characters (kanji are Chinese characters adapted for the Japanese language), which are defined as those in daily use by the Japanese Industrial Standards Committee (www.jisc.go.jp). Chinese makes use of more than 6,000 characters. These numbers are far

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

greater than those of other major languages (for example, there are 26 characters in English, 27 in Spanish, 49 in Hindi, 28 in Arabic, and 33 in Russian). Thus, several years are required for even a professional designer to create a font.

The authors believe that there are two primary obstacles in typographic font generation. Firstly, a wide variation in typographic font styles exists[1-2]. Typographic fonts are composed of distinctive and decorative strokes. These strokes are difficult to extract using existing methods[3-4], owing to their wide shape variations and intersections among strokes. Secondly, a large amount of characters is required for generation[4-5]. We consider that small input is key to facilitating font generation, because the user can quickly make attempts by preparing only several characters.

In order to handle a wide variation of typographic fonts, we propose a novel stroke extraction method using character skeletons. Our key concept is the use of the skeletons, by means of which we can reveal the intersection locations and approximate stroke shapes. Consequently, natural strokes can be extracted.

We capitalize on strokes in a small subset of the target typographic font. Although the strokes in the subset are imperfect for generating the target font, they are sufficient for generating other strokes by applying transformation. The proposed method generates typographic fonts using transformed strokes.

We designed the proposed method to receive samples in an image format. From the perspective of practical scenarios, an ideal generation method must satisfy two requirements: receiving image format samples and a small number of samples. Firstly, we consider the image format. It is more convenient to collect image format samples than vector format ones. For example, when users encounter texts in an unknown style, it is difficult to find samples in a vector format, while the image format is available immediately by capturing photographs of the text. Likewise, using a small number of samples is related to practical use. Collecting numerous samples is a time-consuming task, which may be an obstacle for popularization.

The main contribution of this paper is the development of a method that automatically generates the target typographic font containing thousands of characters from several characters. As illustrated in Figure 1, we generate the complete target font using only its small subset.

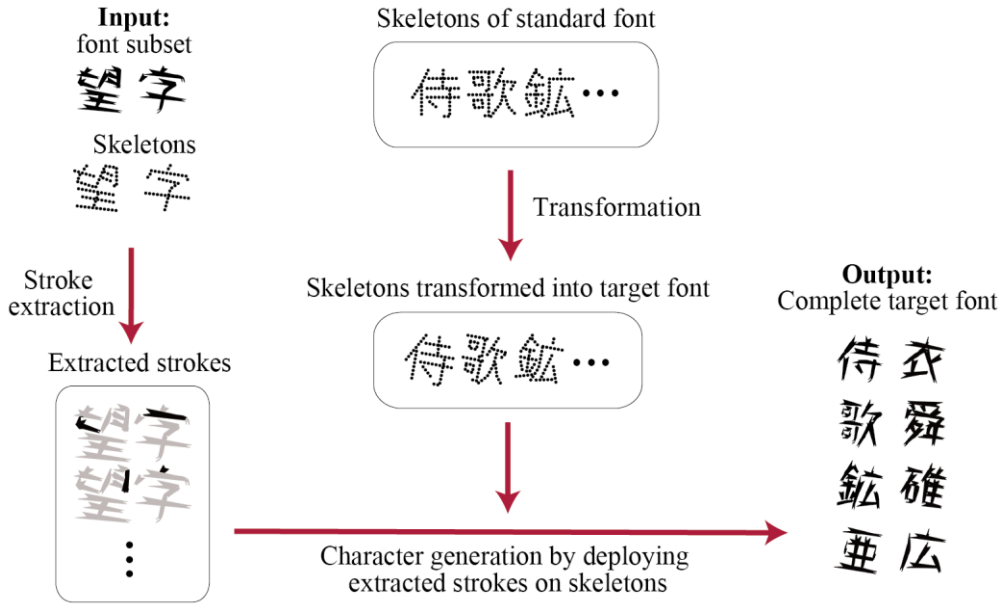
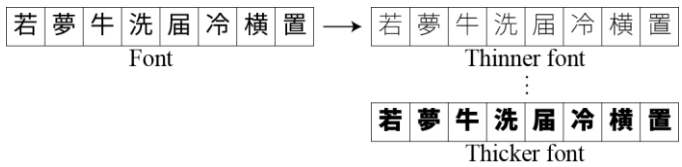


Figure 1. Overview of proposed method. The input is a small subset of the target font images and their skeletons, and the output is the complete target font. We decompose the font subset into strokes using their skeletons. Thereafter, we deploy the strokes to the character skeletons for the target font. We form the skeletons of the target font by transforming skeletons of standard font.

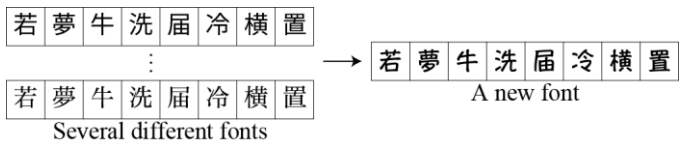
The aforementioned contribution is supported by the experimental results. We used only 15 characters to generate 2,965 characters for a target font. We demonstrated that, with 47 variation in fonts, approximately 140,000 characters could be generated in total. The number of generated characters is sufficiently large, and various fonts are used, including those that are difficult to generate in existing methods owing to being decorated fonts. Moreover, the objective and subjective evaluations verify that the generated characters are similar to the original characters.

RELATED WORK

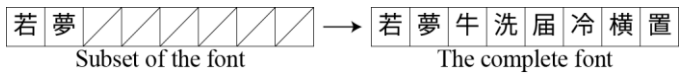
As illustrated in Figure 2, we broadly divided previous studies on font generation into three approaches: *font generation by changing parameters*, *font blending*, and *font generation from a font subset*. We describe these approaches below.



(a) Font generation by changing parameters



(b) Font blending



(c) Font generation from subset

Figure 2. Three approaches in font generation, where the left and right are the input and output, respectively: (a) fonts generated by varying parameters obtained from analysis on given font; (b) new font generated by blending different fonts; and (c) complete font generated from its subset.

Font generation by changing parameters

This approach analyzes a font to obtain parameters so that we can modify the font by varying the parameters, as illustrated in Figure 2 (a). Hu and Hersch analyzed the characters in the Times font family to extract parameters[6]. Character properties such as the thickness and aspect ratio are changed using these parameters. Attempts have been made at analyzing handwritten characters. Djioua and Plamondon used a Sigma-lognormal model supported by the kinematic theory[7]. An interactive system was developed to allow the user to fit a Sigma-lognormal model to alphabetical characters with ease. Wada et al. extracted the trajectories of alphabetical characters and replaced these using a genetic algorithm[8].

The methods addressing this problem generate clean characters. However, they are only applicable to particular fonts that have been analyzed by humans. Furthermore, these methods cannot be applied to Japanese fonts, because over 1,000 characters would need to be parameterized.

Font blending

This approach generates a new font by blending several different fonts; see Figure 2 (b). Su et al. generated Chinese calligraphy characters using a weighted blend of strokes in different styles[9]; they decomposed sample characters into radicals and single strokes, based on rules defined by experts in the field. Suveeranont and Igarashi addressed the generation of alphabetical characters for typographic fonts[2]. They generated characters by blending predefined characters from miscellaneous complete fonts. Campbell and Kautz learned a manifold of standard fonts of alphabetical characters[1]. The locations on the manifold represent a new font. These methods blend fonts and automatically generate fonts. However, obtaining a desired font requires tweaking parameters, and therefore human supervision is required.

In recent years, image generation and style transfer using deep learning have started trending. Lyu et al. transferred calligraphy styles into a standard font using neural networks that were trained with 6,000 character images for a style[10]. The zi2zi project (www.github.com/kaonashi-tyc/zi2zi) generated fonts by means of generative adversarial networks trained with approximately 3,000 characters for every 27 fonts. However, methods based on deep learning require a large dataset for training, which needs to be compiled by a human. In this paper, we focus on font generation from a small subset of the target font to reduce human supervision.

Font generation from subset

This approach generates the complete target font from its subset, as illustrated in Figure 2 (c). The proposed method addresses this approach. Lin et al. generated characters with strokes extracted from given characters using an annotated font, in which the positions and sizes of the strokes were labeled[5]. Stroke extraction was performed on electronic devices so that the characters could be decomposed with ease. Zong and Zhu developed a character-generation method using machine learning; they decomposed the given characters into strokes by analyzing the stroke orientations[3]. The decomposed strokes were assigned to a reference font by means of a similarity function trained by a semi-supervised algorithm. Wang et al. focused on the spatial relationships of strokes for decomposition and generation[4]. Phan et al. extracted stroke outlines by a simple algorithm using skeletons[11]. Lian et al. extracted strokes by comparing target characters with reference characters[12]; the individual strokes in the references were known.

Stroke extraction is a crucial technique for the methods used in this approach. However, most of these methods use naive extraction, which relies on spatial relationships[4] or special devices[5]. It is difficult to extract strokes when their shapes are decorative, or when they are connected or distinctive shapes, such as the Gcomic font (see Figure 8 (a)). However, we extract strokes using character skeletons that clearly specify the stroke locations and relations. Therefore, we can detect strokes from various other strokes, and appropriate extraction strategies can be selected.

To the best of the authors' knowledge, the method developed by Saito et al.[13] is the only one applicable to characters in fonts with various shapes. They applied a patch transform[14] to samples and generated alphabetical characters in a diverse range of fonts. The reader can find details in section 7.3.3 of the book[15]. However, the generated results did not meet the criteria for practical use. In this paper, we propose an adaptive active contour model (AACM) for stroke extraction. Using the proposed method, we can obtain natural character strokes, even in decorated characters.

Other approaches

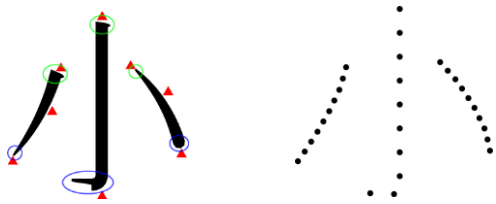
Yang et al. addressed the problem of generating characters by transferring the effects of the target font into binary images of characters[16]. They assigned patches of the effect extracted from a character of the target font to the binary images. They required as many binary images as the characters for generation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

SKELETONS OF STANDARD FONT

Firstly, we address the skeletons of the standard font, as illustrated in Figure 1. We created the skeletons of the standard font based on the GlyphWiki data (www.en.glyphwiki.org).

GlyphWiki is an extremely large database, which contains strokes from more than 280,000 characters in the Mincho style. It includes Chinese, Korean, Japanese, and characters created by users that are not used publicly. Figure 3 (a) illustrates strokes from GlyphWiki. Each stroke in GlyphWiki has control points, a line type, a start shape, and an end shape. The control points guide the trajectory of a stroke. The line type indicates whether the trajectory is a line, curve, or combination of lines and curves. The start and end shapes provide the respective details of the stroke; see blue circles in Figure 3 (a).



(a) GlyphWiki strokes (b) Skeletons of strokes

Figure 3. Visualization of strokes using (a) GlyphWiki and (b) skeletons of standard font. The red triangles represent control points; the green and blue circles represent the start and end shapes, respectively; the black dots in (b) represent points.

The skeletons of the standard font are composed of the points, line type, start shape, and end shape extracted from GlyphWiki. We extract 100 points from the trajectories by regular sampling. The number of points is determined by heuristic analysis. Specifically, we define a skeleton S as

$$S = \left(P, \theta_{\text{line}}, \theta_{\text{start}}, \theta_{\text{end}} \right), \qquad (1)$$

$$P = \begin{bmatrix} \cdots & x_i & \cdots \\ \cdots & y_i & \cdots \\ \cdots & 1 & \cdots \end{bmatrix}, \qquad (2)$$

where P, θ_{line}, θ_{start}, and θ_{end} represent the extracted points, line type, start shape, and end shape, respectively; x_i and y_i represent the x- and y-coordinates of the i th extracted point, respectively. We adopted homogeneous coordinates. The line type θ_{line} indicates whether the line is straight, curved, or a combination of straight and curved.

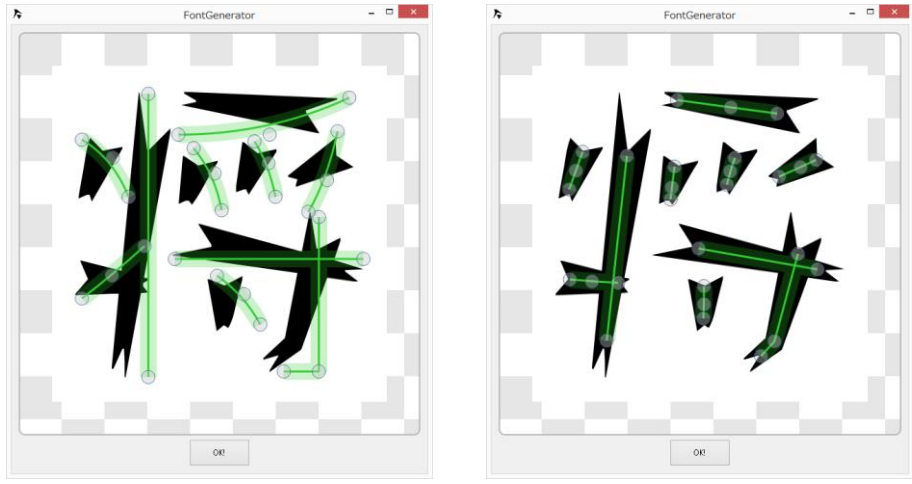
STROKE EXTRACTION

In this section, we address the stroke extraction illustrated in Figure 1. When the proposed method receives a small subset of the target font, strokes are extracted. Firstly, we reveal relationships between strokes that are separate or crossing other strokes. Strokes are extracted by the AACM with adaptive constraints according to these relationships. Finally, we restore the extracted strokes so that natural strokes can be obtained. We carefully designed the stroke extraction so that highly characteristic strokes can be extracted accurately from various fonts, even those that are connected to one another.

Adjusting skeletons to font subset

Stroke extraction is thoroughly developed with the skeletons of the font subset; however, the skeletons of target fonts are generally not available and therefore, we have to create these. Inspired by accurate segmentation applications such as GrabCut[17], we adopt a graphical user interface (GUI) to adjust the skeletons of the standard font to the font subset in order to obtain accurate skeletons of the font subset.

We developed a GUI for adjusting the skeletons of the standard font, as illustrated in Figure 4. The skeletons of the standard font are displayed on the character of the font subset with control points extracted from GlyphWiki data; see Figure 4 (a). The operator adjusts a skeleton by dragging the control points, and the GUI displays these changes. The adjustment result is illustrated in Figure 4 (b).



(a) Before adjustment (b) After adjustment

Figure 4. Skeleton adjustment GUI. Green lines represent skeletons; circles are control points. The skeletons are placed on the font character. (a) Screenshot of GUI initiating an adjustment. Note that the skeleton of the standard font is not matched to the character of the target font at this point, because the skeletons of the standard font differ from those of this font. (b) Adjustment result.

In order to assist the operator, we implemented three automation techniques: scale adjustment, rotation adjustment, and cooperative movement. Firstly, the scale of the skeletons is adjusted to fit the character. We calculate a scaling factor from the rectangles surrounding the skeletons and the character. The second technique is rotation adjustment. We calculate the rotation matrix by matching points of the skeletons of the standard font to points extracted from the medial axis of the character. Thirdly, control points move cooperatively when the start point is moved; for example, if the start point moves up, the other control points also move up. The scale and rotation adjustments are performed only once, prior to manual adjustment.

We emphasize that this adjustment is not difficult, and it can be completed within minutes since the subset only contains 15 characters.

Stroke relationship assignment

Stroke extraction requires special care when strokes are touching or crossing other strokes. Therefore, in this section, we reveal relationships between strokes using skeletons. We define five relationships between two skeletons, as illustrated in Figure 5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

(a) (b) (c) (d) (e)

Figure 5. Relationships between red and black skeletons: (a) continuous, (b) connecting, (c) connected, (d) crossing, and (e) isolated. In these examples, the body part of a skeleton is composed of second and third points, while start and end parts are other points.

- Continuous: the start or end of a skeleton connects to the start or end of another skeleton.
- Connecting: the start or end of a skeleton connects to the body of another skeleton.
- Connected: the body of a skeleton is connected to the start or end of another skeleton.
- Crossing: the skeleton crosses over another skeleton.
- Isolated: the skeleton is isolated from another skeleton.

The `body' is the part of a skeleton other than the start and end parts. We use the function $R(S,S')$ for assigning a relationship to a skeleton *S* against another skeleton *S'*, based on the distance $d(S,S')$ between the two skeletons.

$$R(S,S') = \begin{cases} \text{continuous} & \text{if } d(S,S') < d_{\text{th}} \wedge \neg \mathrm{I}_{\text{body}}(\bar{i}) \wedge \neg \mathrm{I}_{\text{body}}(\bar{j}), \\ \text{connecting} & \text{if } d(S,S') < d_{\text{th}} \wedge \neg \mathrm{I}_{\text{body}}(\bar{i}) \wedge \mathrm{I}_{\text{body}}(\bar{j}), \\ \text{connected} & \text{if } d(S,S') < d_{\text{th}} \wedge \mathrm{I}_{\text{body}}(\bar{i}) \wedge \neg \mathrm{I}_{\text{body}}(\bar{j}), \\ \text{crossing} & \text{if } d(S,S') < d_{\text{th}} \wedge \mathrm{I}_{\text{body}}(\bar{i}) \wedge \mathrm{I}_{\text{body}}(\bar{j}), \\ \text{isolated} & \text{otherwise}, \end{cases} \quad (3)$$

$$d(S,S') = \min_{i,j} \left\| p_i - p_j \right\|_2, \quad (4)$$

where we calculate a threshold d_{th} as $2(\tau+\tau')$, and τ and τ' represent the thickness of *S* and *S'*, respectively. Moreover, $\bar{i}$ and $\bar{j}$ are *i* and *j* minimizing $d(S,S')$. The function I_{body} indicates whether a point p_i is on the body. Let a skeleton have *n* points; I_{body} calculates true or false as follows: $\mathrm{I}_{\text{body}}(\bar{i}) = \text{True} \quad \text{if } 0.05 < \frac{\bar{i}}{n} < 0.95$, otherwise false. The parameters 0.05 and 0.95 were determined experimentally.

Stroke extraction using AACM

At this stage, we have obtained the font subset, skeletons, and their relationships; see Figure 1. We propose an adaptive active contour model (AACM) that can extract strokes even they are crossing by determining the boundary of each stroke using the skeletons and relationships. Inspired by SNAKES[18], the AACM seeks the boundary by optimizing an energy function, namely eq. (5). Moreover, we incorporate constraints and adaptive energy modification into eq. (5) so that strokes can be extracted from complex characters. Examples of stroke extraction using the AACM are presented in Figure 6.

The AACM seeks the boundary by minimizing eq. (5). The boundary is a closed curve, and we represent the position of the boundary by $\mathbf{b}(v) = (x(v), y(v))$. The energy E_{AACM} is defined as

$$E_{\text{AACM}}\left(\mathbf{b}(v)\right) = \int_0^1 \left\{ E_{\text{int}}\left(\mathbf{b}(v)\right) + E_{\text{stroke}}\left(\mathbf{b}(v)\right) \right\} dv, \quad (5)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

where E_{int} and E_{stroke} represent the smoothness of the curves and boundary of a stroke, respectively, while E_{int} follows the work[18].

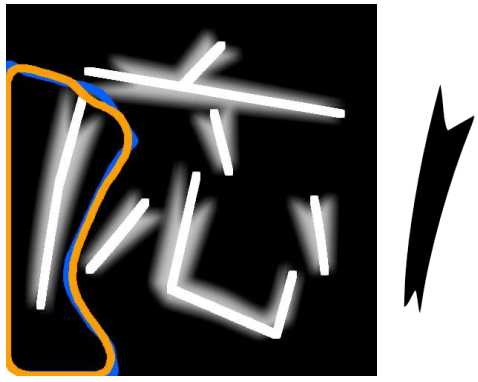
$$E_{\text{int}}\left(\mathbf{b}(v)\right)=\frac{1}{2}\left(\alpha\left\|\mathbf{b}'(v)\right\|^2+\beta\left\|\mathbf{b}''(v)\right\|^2\right),$$

where $\mathbf{b}'$ and $\mathbf{b}''$ represent the first and second derivatives, respectively, and α and β are coefficients. We set α=0.2 and β=0.8. We define E_{stroke} as

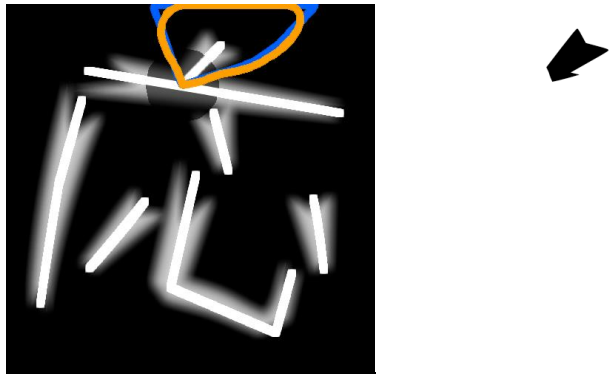
$$E_{\text{stroke}}\left(\mathbf{b}(v)\right)=G_{\text{v}} * I_{\text{char}}\left(\mathbf{b}(v)\right)+I_{\text{sk}}\left(\mathbf{b}(v)\right), \quad (6)$$

where G_v is a Gaussian kernel with variance v, $*$ is convolution operation, while I_{char} and I_{sk} are the gray-scale images of the character and skeletons. The pixel values are 255 if they are characters or skeletons, and 0 otherwise. The pixel value represents the energy: 0 is the lowest and 255 is the highest. Therefore, E_{stroke} has high energy near the skeletons.

For the purposes of this study, it is unnecessary for a boundary to be very close to its stroke, and instead, it may be rather relaxed. If the boundary is too close, it may cross the stroke, and consequently, the extracted strokes will be damaged.



(a) Extraction of isolated stroke



(b) Extraction of connecting stroke

Figure 6. Examples of stroke extraction. The boundaries of AACM on energy E_{stroke} and extracted stroke are shown on the left and right, respectively. The initial AACM (blue) and minimized AACM (orange) are illustrated. The black and white colors represent the energy, where black is the lowest and white is the highest.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

We classify all pixels of I_{sk} to the nearest skeleton. We use the boundaries of the segmentation results as the initial boundary of the AACM. Moreover, we adaptively modify E_{stroke} and set several points through which the boundary must pass.

When the target stroke is continuous, connecting, connected, or crossing, we adaptively modify E_{stroke}, and set points through which the AACM must pass so that the AACM can encompass only the target stroke.

In cases where the target stroke relationship is connecting or continuous, we modify E_{stroke} and constrain the AACM to go through the start or end point of the skeleton, which is near to another skeleton. As illustrated in Figure 6 (b), E_{stroke} is decreased around the intersection, and the AACM is constrained to pass through the point. Consequently, the boundary including only the target stroke is obtained. In particular, we decrease E_{stroke} by $1/\beta_1$ within the distance $\beta_2\tau$ from the point, except for I_{sk}, where β_1 and β_2 are constant values that were determined empirically.

In cases where the target stroke relationship is crossing or connected, we decrease E_{stroke} by $1/\beta_1$ within distances from other strokes of $\beta_2\tau_{others}$. Thereafter, we minimize the AACM with the modified E_{stroke} and extract the target along the minimized AACM.

Moreover, we developed a method for stroke restoration so that the natural strokes could be extracted. Kindly refer to supplemental materials for further details.

CHARACTER GENERATION

In this section, we address the character generation illustrated in Figure 1. The generation is fully automatic, and is carried out once the stroke extraction is completed. The proposed method consists of two processes. During the first process, we take the skeletons of the standard font and transform them into the target font by applying a transformation matrix, which is estimated from the skeletons of the font subset. Subsequently, we generate characters by deploying the extracted strokes on the transformed skeletons.

Skeleton transformation into target font

The results of the character generation would be odd in appearance—even with perfect strokes—if we used the skeletons of the standard font, because they could differ from the target font. Thus, we transform the skeletons of the standard font so that we can obtain skeletons that mimic the target font. It should be noted that the transformation is not intended to change the structure of the characters. This transformation is performed to adjust the character skeletons to a unified size and geometry.

We estimate the transformation matrix from the skeletons of the font subset. In particular, we seek two transformation matrices: T_{sz} and T_{aff}, where T_{sz} adjusts the size of the skeletons and centroid translation, while T_{aff} adjusts the affine transformation of the skeletons. Transformation is achieved by applying T_{sz} and T_{aff} to the skeletons of the standard font. Specifically, we adjust the sizes, centroids, shear, and rotations. Note that these transformations can be represented by affine transformations.

We estimate T_{sz} from the skeletons of both the standard font and font subset. Let the font subset be a set of characters $\boldsymbol{C} = \{\cdots, c_i, \cdots\}$, where H_i and W_i denote the height and width of the skeletons of the standard font for c_i, respectively. Likewise, $\hat{H}_i$ and $\hat{W}_i$ represent the height and width of the skeletons of the font subset for c_i, respectively. With an output image size of W_{img} and H_{img}, we can calculate T_{sz}, as follows:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

$$T_{sz} = \frac{1}{|\mathbf{C}|}\sum_{c_i \in \mathbf{C}} \begin{pmatrix} \hat{W}_i / W_i & 0 & W_{img}\dfrac{\hat{W}_i}{2W_i} \\ 0 & \hat{H}_i / H_i & H_{img}\dfrac{\hat{H}_i}{2H_i} \\ 0 & 0 & 1 \end{pmatrix}. \quad (7)$$

We estimate T_{aff} using transformations from the standard font skeletons to the font subset skeletons. However, T_{aff} would become a trivial matrix, such as an identity matrix, because certain transformations cancel one another. In particular, these canceling transformations occur in curved skeletons or skeletons in different radicals. To avoid such problems, we only use straight skeletons. Moreover, we calculate the transformations for groups consisting of skeletons whose relationships are continuous, connecting, connected, or crossing with another skeleton in the group. If a skeleton is isolated from all skeletons in the group, it belongs to another group. Consequently, we can avoid cancelations in different radicals. Suppose that we have N_{group} groups over skeletons of the font subset, and calculate a transformation T_i for a group,

$$T_{aff} = \frac{1}{N_{group}}\sum_{i=1}^{N_{group}} T_i , \quad (8)$$

$$T_i = \sum_{P \in S_{line}} \arg\min_{T} \left\| \hat{P} - TP \right\|_2 , \quad (9)$$

where S_{line} represents the straight skeletons of the standard font in a group, and $\hat{P}$ is a set of points of the skeleton of the font subset. We calculate T_{aff} by averaging the affine transformations over the groups. In practice, the transformations of groups hardly cancel one another, as rotations are consistent within a font.

Stroke deployment

We select the most suitable stroke for the skeleton of target character from the extracted strokes. Subsequently, we transform the found extracted stroke to fit to the skeleton. We repeat this selection until extracted strokes are assigned to all skeletons of the target character.

Suitable strokes must be selected, because they affect the appearance of the generated characters. The challenge in this case is to measure the suitability of a stroke for a skeleton. As the strokes are images, while the skeletons are points and parameters, they cannot be compared directly. Therefore, we focus on the skeletons of the extracted strokes, that is, the skeleton of the font subset, to develop suitability in terms of the following aspects:

- The suitable stroke has a skeleton that fits the skeleton of the target font by means of a transformation.
- A small transformation is preferable; a large transformation will distort the stroke.
- The line type, start shape, and end shape of the skeleton of the suitable stroke are the same as those of the skeleton of the target font. This facilitates strokes being deployed on skeletons that have the same line type. For example, straight strokes are deployed on straight skeletons, and curved strokes on curved skeletons.

In particular, we select the closest stroke according to distance function f_{d}, which calculates the distance using skeletons. We compare skeleton S of the font subset with skeleton S' of the target font, which is transformed by applying T_{sz} and T_{aff} to skeletons of the standard font. Please

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

refer to eq. (1) for details regarding *S*. The skeleton of the standard font $\hat{S}$ is compared with the skeleton of the standard font $\hat{S}'$, and these are associated with *S* and *S'*, respectively.

We define $f_d(S, S')$ as:

$$f_d(S,S') = d_{\text{stroke}}(S,S') + d_{\text{stroke}}(\hat{S},\hat{S}') + d_{\text{neighbor}}(\hat{S},\hat{S}'), \quad (10)$$

$$d_{\text{stroke}}(S,S') = \frac{1}{N}\left(\left\|P' - TP\right\|_1 + \left\|P - TP\right\|_1\right) - \mathrm{I}(\theta_{\text{start}},\theta'_{\text{start}}) - \mathrm{I}(\theta_{\text{end}},\theta'_{\text{end}}), \quad (11)$$

$$d_{\text{neighbor}}(\hat{S},\hat{S}') = \begin{cases} \left\|\hat{S}_{\text{st}} - \hat{S}'_{\text{st}}\right\|_1 + \left\|\hat{S}_{\text{ed}} - \hat{S}'_{\text{ed}}\right\|_1 & \text{if } \hat{S}_{\text{st}}, \hat{S}_{\text{ed}}, \hat{S}'_{\text{st}}, \text{and } \hat{S}'_{\text{ed}} \text{ exist} \\ 50 & \text{otherwise} \end{cases}, \quad (12)$$

where *N* is the number of points of *P*, and *T* represents the transformation from *P* to *P'*. We obtain *T* by means of the least-squares method: $T = (P^{\mathrm{T}}P)^{-1}P^{\mathrm{T}}P'$. Here, I is 1 if the two parameters are the same, otherwise it is 0. Moreover, $\hat{S}_{\text{st}}$ represents the points of a skeleton connected to the start part of $\hat{S}$. In the case where several skeletons are connected to $\hat{S}$ at the start part, we select one skeleton, the center of which is closest to $\hat{S}$. Likewise, $\hat{S}_{\text{ed}}$ represents a skeleton connected to the end part of $\hat{S}$. If $\hat{S}_{\text{st}}$, $\hat{S}_{\text{ed}}$, $\hat{S}'$, or $\hat{S}'_{\text{ed}}$ is lacking, d_{neighbor} will be the constant value.

When the suitable stroke is identified, we deploy it by applying the transformation from the skeleton of the stroke to the skeleton of the target font.

EXPERIMENTAL RESULTS

We demonstrate font generation using the proposed method. We used five target fonts: Ibara, Gcomic, Onryo, Tsunoda, and Zinpen (see supplemental materials). Figure 7 (a) illustrates the original characters in the font subset for the five fonts. We selected these characters using the sample selection method described in the supplemental materials. These characters contain all of the parameters for θ_{line}, θ_{start}, and θ_{end}. We generated 2,965 characters in which a set of JIS level-1 exists, using $\beta_1 = 2.0$ and $\beta_2 = 1.5$.

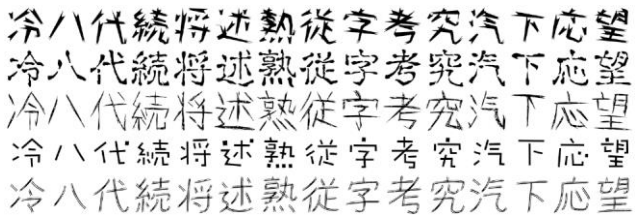
For comparison, we also used the Saito method[13]. To the best of the authors' knowledge, this is the only method applicable to characters in various typographic fonts. This Saito method applies patch transform[14], which breaks an image into small patches and generates a modified image by rearranging these under a certain constraint. The target of this method is a natural image. The arrangement is defined as an optimization problem, considering that there is no unnaturalness or inconsistency in an image. Based on this concept, the Saito method generates a character image by breaking the provided font patterns into small patches and rearranging them. As character images are binary, a problem exists whereby the continuity of an image is lost near the boundary of the patches and it easily becomes unnatural, which is an issue that should be resolved. The frameworks of the proposed and Saito methods are similar. Both exploit character images, extract character components, and generate characters. However, the extracted components differ significantly. The components of the Saito method are small pieces of characters that lack meaning, while the components of the proposed method are complete strokes.

Character generation results

The characters generated in the five fonts using the proposed method and Saito method are illustrated in Figure 7. The characters generated by the proposed method appear clean and exhibit strong readability. Moreover, the font characteristics, such as the slant of the characters in Ibara

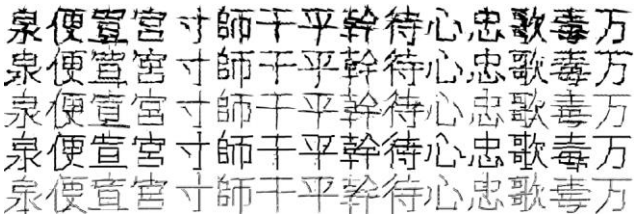
and the size in Tsunoda, are successfully reconstructed. These results support the effectiveness of the skeleton transformation using affine transformations.

Although results of only 15 characters are presented in Figure 7 owing to space limitations, as mentioned previously, we generated 2,965 characters composed of simple and complex characters. The stroke complexity statistics for the generated characters are: 1 (min), 29 (max), 12.5 (avg), and 4.5 (std).



(a) Input subsets

(b) Original font



(c) Results of Saito method

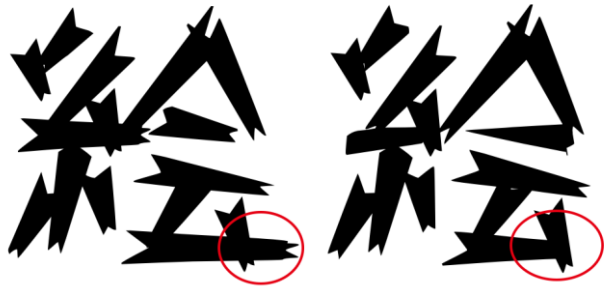
(d) Result of proposed method

Figure 7. Input subsets and results. From top to bottom: Ibara, Gcomic, Onryo, Tsunoda, and Zinpen.

We conducted an experiment to investigate the effect of the number of sample characters in the font subset. In particular, we changed the number of sample characters in the subset and generated characters. A smaller subset contained a lower number of strokes; for example, there were five strokes in two characters, but 10 strokes in three characters. The number of generated characters reached 2,965 at 13 characters or above. Note that the maximum was 2,965. The number of generated characters was dependent on the stroke parameters θ_{line}, θ_{start}, and θ_{end} in the subset,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

because all parameters were required to generate all characters. In fact, we generated only 1,012 characters at 2, and 2,965 characters at 13. According to this result, the smallest subset that generated all characters was the subset of 13 characters. We consider that redundancy of the stroke parameters may improve the quality of the results. An example is presented in Figure 8. Comparing the characters generated using subsets including 13 and 15 characters, the latter character is preferred. The difference is denoted by red circles.



(a) 13 characters (b) 15 characters

Figure 8. Example of generated characters improved by redundancy of stroke parameters.

Subjective evaluation

In this section, we present the results of the subjective evaluation regarding the consistency in the fonts. The subjective evaluations were performed using 14 participants.

The participants evaluated the consistency between the styles of the generated characters and original fonts. In the evaluation, we showed the original characters in Figure 7 (a) to the participants so that they could learn the fonts. Thereafter, 160 idioms consisting of four characters were displayed. The idioms were written in the original characters, characters generated by the proposed method or characters generated by the Saito method. We placed the mean opinion scores (MOSs) over the idioms. The participants assigned scores ranging from 1 (bad) to 5 (excellent) to the idioms, according to their impressions of the font consistency. Table 1 summarizes the results. The proposed method received higher MOSs than the Saito method. Moreover, the MOSs for the proposed method were relatively close to those for the original characters.

Table 1. MOSs for font consistency. The score ranged from 1 (bad) to 5 (excellent)

	Ibara	Gcomic	Onryo	Tsunoda	Zinpen	Ave.
Original	4.5	4.8	4.5	4.9	4.9	4.7
Saito	1.1	1.3	1.5	1.1	1.3	1.3
Proposed	4.3	4.4	4.6	4.3	4.6	4.4

Varied font generation

To demonstrate the font generation capability of the proposed method, we performed a generation experiment with 42 fonts (see supplemental materials): four standard fonts (1 to 4), six calligraphy handwriting fonts (5 to 10), 17 pen handwriting fonts (11 to 27), and 15 artificial fonts (28 to 42). The fonts were varied and included Mincho, Gothic, clerical, antique, personal handwriting, professional, and handwriting styles. We generated 2,965 characters for each font and illustrate several of the results in Figure 9. The results are promising: the proposed method generated clean characters and reproduced the style of each font.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2019.2931431, IEEE Computer Graphics and Applications

(a) Original (b) Generated

Figure 9. Examples of generated characters in 42 fonts.

CONCLUSION

In this paper, we have proposed a method to generate a typographic font from a small subset of the target font. The proposed method successfully extracts the natural strokes from character images, and constructs characters by deploying the appropriate strokes onto skeletons that are generated automatically. The proposed method can serve as a valuable aid for font designers, because they only have to create several characters; other characters will be generated automatically.

An experimental evaluation was conducted using five characteristic fonts that are difficult to generate by means of existing methods. We evaluated the generated characters using both objective and subjective tests; all of the results indicate that the characters generated by the proposed method exhibited appearances, usefulness, and readability comparable to the original characters. Furthermore, we used 42 additional fonts to demonstrate the generative capabilities of the proposed method. Therefore, we generated 2,965 characters for 47 fonts, resulting in a total of 139,355 characters. Moreover, we only used 15 characters for each font in the generation.

In future work, we will attempt to adjust skeletons automatically using other datasets containing stroke information to improve the quality of the generated character images. An important issue is skeleton transformation for fonts that cannot be mimicked using affine transformations. We will study the structural changes of the characters, so that the generation can be flexible. Moreover, we will develop a more accurate evaluation method focusing on strokes.

ACKNOWLEDGEMENTS

This work was partially supported by JST, CREST, and JSPS KAKENHI, under grant numbers 16K00259 and 19K12033.

REFERENCES

1. N. D. F. Campbell and J. Kautz, "Learning a Manifold of Fonts," *ACM SIGGRAPH*, 33, 4, 2014.
2. R. Suveeranont and T. Igarashi, "Example-Based Automatic Font Generation," *Smart Graphics*, pp.127-138, 2010.
3. A. Zong and Y. Zhu, "StrokeBank: Automating Personalized Chinese Handwriting Generation," *Proceedings of the Twenty-Sixth Annual Conference on Innovative Applications of Artificial Intelligence*, pp. 3024-3029, 2014.
4. Y. Wang, H. Wang, C. Pan and L. Fang, "Style Preserving Chinese Character Synthesis Based on Hierarchical Representation of Character," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.1097-1100, 2008.
5. J. W. Lin, C. Y. Hong, R. I. Chang, Y. C. Wang, S. Y. Lin and J. M. Ho, "Complete Font Generation of Chinese Characters in Personal Handwriting Style," *IEEE International Performance Computing and Communications Conference*, pp. 1-5, 2015.
6. C. Hu and R. D. Hersch, "Parameterizable Fonts Based on Shape Components," *Computer Graphics and Applications*, 21, 3, pp. 70-85, 2001.
7. M. Djioua and R. Plamondon, "An Interactive System for the Automatic Generation of Huge Handwriting Databases from a Few Specimens," *International Conference on Pattern Recognition*, pp. 1-4, 2008.
8. Y. Wada, H. Kasuga and K. Sumita, "An Evolutionary Approach for the Generation of Diversiform Characters using a Handwriting Model," *International Conference on Pattern Recognition*, 3, pp. 131-134, 2002.
9. S. Xu, F. Lau, W. K. Cheung and Y. Pan, "Automatic Generation of Artistic Chinese Calligraphy," *IEEE Intelligent Systems*, 20, 3, pp. 32-39, 2005.
10. P. Lyu, X. Bai, C. Yao, Z. Zhu, T. Huang and W. Liu, "Auto-Encoder Guided GAN for Chinese Calligraphy Synthesis," *International Conference on Document Analysis and Recognition*, pp. 1095-1100, 2017.
11. H. Q. Phan, H. Fu and A. B. Chan, "FlexyFont: Learning Transferring Rules for Flexible Typeface Synthesis," *Pacific Graphics Forum*, 34, 7, pp. 245-256, 2015.
12. Z. Lian, B. Zhao and J. Xiao, "Automatic Generation of Large-scale Handwriting Fonts via Style Learning," *SIGGRAPH ASIA Technical Briefs*, 12, 4, pp. 1-4, 2016.
13. H. Saito, Y. Sugaya, S. Omachi, S. Uchida, M. Iwamura and K. Kise, "Generation of Character Patterns from Sample Character Images," *IEICE Technical Report. PRMU*, 110, 467, 2011.
14. T. S. Cho, M. Butman, S. Avidan and W. T. Freeman, "The Patch Transform and Its Applications to Image Editing," *IEEE Computer Vision and Pattern Analysis*, pp. 1-8, 2008.
15. T. Nishida, "Human-Harmonized Information Technology, Volume 2: Horizontal Expansion," *Springer Japan*, pp. 197-233, 2017.
16. S. Yang, J. Liu, Z. Lian and Z. Guo, "Awesome Typography: Statistics-Based Text Effects Transfer," *IEEE International Conference on Computer Vision*, 2017.
17. C. Rother, V. Kolmogorov and A. Blake, "GrabCut: Interactive Foreground Extraction Using Iterated Graph Cuts," *ACM Trans. Graph*., 23, 3, pp. 309-314, 2004.
18. M. Kass, A. Witkin and D. Terzopoulos, "Snakes: Active Contour Models," *Int. journal of computer vision*, 1, 4, pp. 321-331, 1988.

ABOUT THE AUTHORS

Tomo Miyazaki is an Assistant Professor of the Graduate School of Engineering, Tohoku University. Contact him at tomo@iic.ecei.tohoku.ac.jp.

Tatsunori Tsuchiya was a graduate student in master's degree at the Graduate School of Engineering, Tohoku University, before joining Canon Inc.

Yoshihiro Sugaya is an Associate Professor of the Graduate School of Engineering, Tohoku University. Contact him at sugaya@iic.ecei.tohoku.ac.jp.

Shinichiro Omachi is a Professor of the Graduate School of Engineering, Tohoku University. Contact him at machi@ecei.tohoku.ac.jp.

Masakazu Iwamura is an Associate Professor of the Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University. Contact him at masa@cs.osakafu-u.ac.jp.

Seiichi Uchida is a Professor of the Faculty of Information Science and Electrical Engineering at Kyushu University. Contact him at uchida@ait.kyushu-u.ac.jp.

Koichi Kise is a Professor of the Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University. Contact him at kise@cs.osakafu-u.ac.jp.