Pacific Graphics 2015
N. J. Mitra, J. Stam, and K. Xu
(Guest Editors)

Volume 34 (2015), Number 7

FlexyFont: Learning Transferring Rules for Flexible Typeface Synthesis

H. Q. Phan[1] and H. Fu[1] and A. B. Chan[2]

[1]School of Creative Media, City University of Hong Kong
[2]Department of Computer Science, City University of Hong Kong

Abstract

Maintaining consistent styles across glyphs is an arduous task in typeface design. In this work we introduce FlexyFont, a flexible tool for synthesizing a complete typeface that has a consistent style with a given small set of glyphs. Motivated by a key fact that typeface designers often maintain a library of glyph parts to achieve a consistent typeface, we intend to learn part consistency between glyphs of different characters across typefaces. We take a part assembling approach by firstly decomposing the given glyphs into semantic parts and then assembling them according to learned sets of transferring rules to reconstruct the missing glyphs. To maintain style consistency, we represent the style of a font as a vector of pairwise part similarities. By learning a distribution over these feature vectors, we are able to predict the style of a novel typeface given only a few examples. We utilize a popular machine learning method as well as retrieval-based methods to quantitatively assess the performance of our feature vector, resulting in favorable results. We also present an intuitive interface that allows users to interactively create novel typefaces with ease. The synthesized fonts can be directly used in real-world design.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Typeface design is an initial stage in font design, where individual glyphs are created and put together under a typeface. It requires dedicated skills and is still a laborious process even for professionals. One of the difficulties is to maintain consistent styles across glyphs. Because the number of glyphs is large even for a minimalist alphabetic system like Latin alphabet, which is the focus of our work, it often takes weeks to design a complete consistent typeface from scratch. During the design process, designers are encouraged to maintain a library of glyph parts, which can be reused to achieve consistency within a typeface and between typefaces in the same family [Bos12]. See an example of such parts in Fig. 1. Hence, a glyph can be naturally represented as a composition of a small number of properly arranged parts.

Recently, the interest in font research has increased in computer graphics communities. A common approach is to perform outline interpolation from exemplar glyphs of the same characters. For example, Suveeranont and Igarashi [SI10] automatically generate complete typefaces from a single example by computing blending weights for interpolating between glyphs of the same characters in an existing font library. The "blending weights" can be considered as a representation of typographic style. Similarly, the work of [CK14] achieves outline interpolation by embedding a few input glyphs, which should be similar to the fonts in the dataset, into a low-dimensional latent space learnt from the corresponding glyph outlines of the same characters and then projecting back to the original space to get a complete typeface. The assumption that a glyph is an average of other same-character glyphs is restrictive, and does not allow for glyph styles that are very different from those present in the font library. A natural limitation of those approaches is that

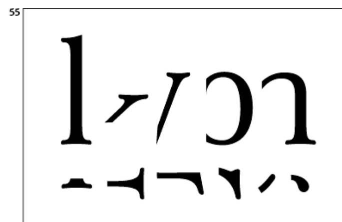

Figure 1: *Maintaining a library of parts helps keep the consistency in typefaces [Bos12].*

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

Figure 2: *Varieties of font styles, which are characterized by the repetitions of caps (Top) and/or brush types (Bottom).*

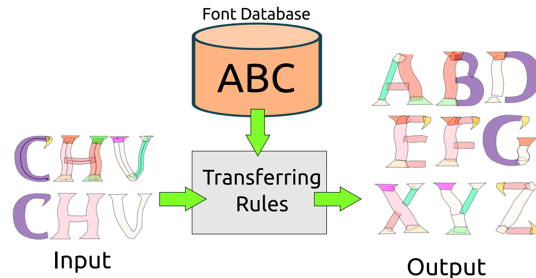

Figure 3: *Given a few input glyphs, our part-assembly approach infers transferring rules using a pre-learned statistical model from a font database, and then synthesizes missing glyphs by assembling the parts of the given input glyphs using the inferred transferring rules.*

they do not take into account the style intended by the designer, e.g., by providing several designed glyphs.

We observed that the repetitive and consistent usage of parts among glyphs, especially for decorative fonts that do not conform to a standard typographical rule, can be considered as the "style" for a specific typeface. For example, Fig. 2 shows varieties of font styles, which are mainly characterized by the repetitive use of parts in different characters. Fig. 2 (Top) shows the "serif" styles while the bottom part exhibits the "stroke" styles. Note the differences in part repetition in each font. For instance, the "Playball" and "Supermercado" fonts only have serifs on top of the glyphs while "Cherry Swash" has them on both top and bottom. Standard fonts like "Times" have the same type of serif across glyphs, while "Cherry Swash" has 2 types of serifs, one for the left side and the other for the right side. Similarly, there are differences in stroke types for the fonts at the bottom of Fig. 2. It is important to note that, "styles" or part repetitions are sometimes mixed together. For instance, the "Limelight" stroke style is a mix of "Cinzel Deco." and "Geostar Fill". These observations suggest that using a fixed set of correspondences between parts and glyphs is insufficient to represent the wide variety of styles of decorative fonts.

In this paper we propose a framework that, given one or more outline-based glyphs of several characters as input, produces a complete typeface which bears a similar style to the inputs, as illustrated in Fig. 3. Unlike previous interpolation-based approaches [SI10, CK14], our method takes a *part assembly* approach by first decomposing the input glyphs into semantic parts and then inferring the transferring rules for the complete typeface from these parts. The missing glyphs are then reconstructed by assembling the parts according to the predicted rules. The advantage of our approach is that the style given by the designer is well preserved while the transferring rules help keep consistency in style across the synthesized typeface.

Specifically, we represent typographic style as the similarities between glyph parts of different characters in a typeface. This problem has not been studied before. To this end, we first decompose input glyphs into semantic parts (Fig. 3 (left)). A glyph part is represented as a brush stroke that follows a certain skeleton. The brush stroke itself has a brush type and two optional caps at the terminals. Thus, the word "style" can refer to the brush, cap or skeleton style. This representation makes it convenient to reuse the given parts for reconstructing the missing glyphs. Next, the pairwise similarities between all parts of a font are computed and stored in a single vector, which we call the *Part-Similarity Vector* (PSV). All the PSVs extracted from a font dataset are used for learning and interpolating typographic styles. We use a popular distribution modelling method called Bayesian Gaussian Process Latent Variable Model (BGPLVM) as well as simple retrieval based methods to assess the performance of our font style representation. Additionally, we also design an intuitive user interface for font design that allows a designer to interactively incrementally design a complete typeface from scratch. Both qualitative and quantitative results are shown in Sections 5 and 6, respectively.

Terminology In the following sections we refer to *glyph* as a concrete design of a character. One character may have multiple glyphs in different fonts. A *typeface* is a collection of glyphs which usually has a consistent look and feel. A *font* is a medium to store typefaces in computer. We use *font* and *typeface* interchangeably in this paper. We also use the word *stroke* to refer to a polygon created by drawing a brush along a trajectory. A *stroke* may have *caps* which reside at the two terminals of the trajectory. *Part* is used to refer to both *stroke* and *cap*.

2. Related Work

Font Synthesis. Many efforts have been put into designing methods for font synthesization. Early works utilized parametric approaches which parameterize glyphs with a fixed number of parameters, thus allowing users to create variations of an existing font by changing these parameters. Donald Knuth was the first to introduce a working parametric font system and is still being used in modern font design [Knu79]. Shamir and Rappoport [SR98] proposed a higher-level font parametrization method. Well-defined glyph parts and their constraints are taken into account, allowing users to change the appearances of individual parts. The definition of parts in this work is similar to ours in

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

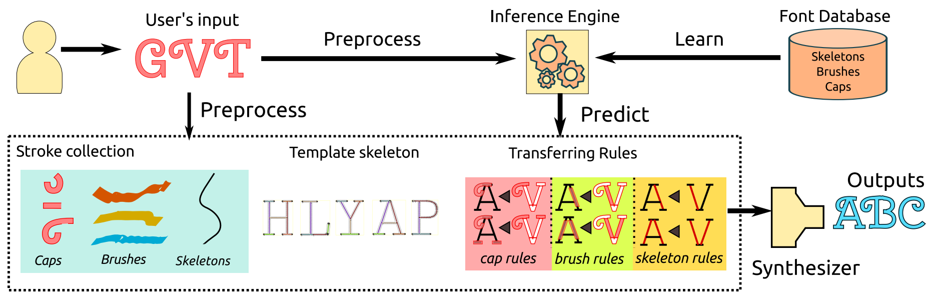

Figure 4: *Overview of our technique.*

the sense that glyph parts have semantic interpretation. In contrast, the work of Hu and Hersch [HH01] focuses more on shape-based parts instead of semantic parts. Parametric methods are useful to create shape variations of an existing font. However, it is difficult to use them for typeface design from scratch.

The idea of interpolating parameters from examples was initially introduced by Knuth [Knu79] and later extended to Chinese characters by Xu and colleagues [XLCP05]. More discussions on Knuth's works on font systems can be found in Hofstadter's book [Hof85]. Lau [Lau09] introduced a parametric model in which parameters are also computed from examples. However, the learning method and the parameters are rigid and allow limited ways of customization like stroke weight or glyph size. In contrast, our model is designed to be flexible, allowing generation of novel typographic styles by mixing the learned transferring styles instead of the examples.

Suveeranont and Igarashi [SI10] extended Xu et al.'s method [XLCP05] to automatically generate Latin glyphs from a single glyph of a character (e.g., 'A'). The input glyph is used to compute mixing weights, which tell how much a certain font in the database contributes to the given glyph. The weights are then used to synthesize the glyphs of other characters (e.g., 'B', 'C') by interpolating between the outlines of the same characters in different database fonts. Their objective is very similar to ours. However, we take a very different approach, including glyph representation and synthesization methods. While their implementation also involves parts, such parts are mainly used as intermediate representation for glyph interpolation, instead of part assembly in our approach. A natural limitation of outline-based interpolation methods is that the quality of synthesized results depends heavily on the font database, since their synthesized results are essentially averaged database typefaces. In an extreme case where there is only one typeface in the database, interpolation-based methods could not produce any variation, while a part-assembly approach is still able to produce different results with respect to different input glyphs even using a fixed set of transferring rules extracted from the single database typeface. Fig. 10 (top) demonstrates this idea.

Campbell and Kautz [CK14] recently introduced a new outline-based interpolation method for font synthesis by learning a low-dimensional glyph outline manifold from data. This manifold is useful for font exploration and sampling of smooth in-between glyphs. However, the sampled fonts are not actually "new" fonts, since their approach has very limited ability to extrapolate new fonts from this manifold, as admitted by the authors. In addition, the learning method depends heavily on point-wise correspondences between glyphs of the same characters in different fonts, making their interpolation-based method only work with standard fonts. Our part assembly method, however, is designed to work with exotic fonts and is able to construct completely novel-looking fonts respecting the user-designed input glyphs. O'Donovan et al. [OLAH14] introduced an interesting interface for font browsing and retrieval.

Stroke Extraction and Stylization. Another field related to our work is stroke extraction and stylization. Zitnick [Zit13] tackled the task of handwriting beautification by firstly decomposing and matching strokes across glyphs, and then obtaining smooth strokes by averaging corresponding strokes (i.e., an interpolation-based approach). Their matching method based on curvature is similar to our skeleton segmentation part in spirit. However, our method does not use any temporal information, and works with typefaces rather than handwriting.

A more generic work of [LYFD12] introduced an efficient way to stylize users' strokes with similar strokes retrieved from a high quality database. Input strokes are also broken into smaller parts which match with strokes in the database. After matching, both strokes and trajectories can be adjusted to have better appearance. The idea of querying and matching strokes is a common point with our work. Nevertheless, simply taking strokes from database does not work for the case of typeface synthesis, since stroke composition in our problem must follow certain typographic rules. Besides, our focus is on semi-professional designers who have their own ideas but have little experience of typeface design. The idea of part assembly has been explored extensively for 3D modeling (e.g., [KLM*13, KCKK12]). Our method can be categorized into this approach, though we only consider the spe-

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

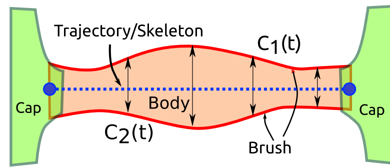

Figure 5: *Stroke-based glyph representation.*

cific domain of font synthesis. Ma and colleagues [MHS*14] introduced an interesting approach to 3D style transfer by analogizing shapes from source to target.

3. System Overview

Our method takes one or more vector-based glyph outlines as input, which implicitly define a designer's intended style. It then produces a complete typeface, which bears the same style as the input. To achieve this, we first decompose the input glyphs into semantic parts, and then assemble them according to some "copying rules" predicted from the input. Figure 4 gives an overview of our framework, which is divided into three steps:

1. **Pre-processing**: The goal of the pre-processing step is to transform an outline-based glyph into a representation suitable for font synthesis and feature extraction. The pre-processing step is applied to both the fonts used for training, as well as the user's input glyphs for synthesization. For each glyph, the outline-based representation is converted to simple polygons. Next, the glyphs are decomposed into semantic parts. The glyphs are then represented as stroke collection, which contains "brushes", "caps" and "skeletons".
2. **Learning and Inference:** The purpose of the learning step is to capture the glyph composition rules and to predict new transferring rules for the typeface to be synthesized. For each font in the dataset, part-similarity vectors (PSVs) for "brush", "cap" and "skeleton" are constructed to represent the style of the font. The distribution of PSVs can then be learned using generative probabilistic models, such as LDA, GPLVM, or GMM. Fig. 6 gives an overview of our style learning process. The goal of the inference step is to correctly infer the style for the entire typeface. Partially-observed PSVs are extracted from the input glyphs, and the missing elements of the vectors are inferred using the learned probabilistic model.
3. **Synthesization:** Given the complete PSVs, the missing glyphs are synthesized by picking appropriate elements from the vectors, and then assembling the parts of the input glyphs accordingly. The synthesization step is carried out heuristically by first copying the thickness values along a skeleton from the source to the destination, and then arranging the "caps" accordingly. The skeleton itself can be obtained through retrieval and/or template deformation.

4. Methodology

Now we give the details of each step.

4.1. Pre-processing

Stroke-based Glyph Representation. To prepare for the synthesization and learning steps, we first represent glyphs as compositions of brush strokes painted along some skeletons. This type of stroke-based glyph representation (SBR) has been used to represent Chinese logograms as they are natural compositions of well defined strokes [XLCP05, JPF06]. Although Latin characters were originally composed by brush strokes, the development of printing technologies resulted in typefaces in which it is more difficult to identify the strokes. Bold typefaces, for instance, have relatively vague stroke borders. However, SBR is very compact and has great flexibility, which suits well to our purpose. We adopt a stroke-based representation similar to stylized stroke fonts [JPF06]. The main difference is we use a pair of parametric curves to represent the two sides of a brush, thus affording richer brush shapes. Our SBR consists of brushes and caps. As illustrated in Fig. 5 for a simple glyph ('I'), a brush defines the shape of a stroke by specifying the thickness along its skeleton, while a cap is a rigid polygon, which is stationed at a stroke's terminals.

As most of the available fonts are outline-based [JPF06], we need to convert a glyph from an outline-based representation to SBR. The stroke extraction process is performed once over the training fonts and is used again at test time when novel input glyphs are given. We first convert glyphs to polygons and extract curve-skeletons of the polygons by using a skeletonization technique [AAAG96]. We (for the database fonts) or a user (for the input glyphs) then manually annotates the two terminals of the body ("trunk") of each desired stroke in each glyph, by clicking on two vertices on the extracted curve-skeleton (blue dotted line in Fig.5). Each pair of such terminals determines a path and thus the trajectory/skeleton of a stroke. Given each trajectory path, we use a simple cross-section heuristic to find the associated vertices on the glyph outline and use these vertices to define the body of a stroke. Once all the stroke trunks are determined, we identify the remaining portions of the curve-skeleton as caps.

Brush Reconstruction. The motivation of using brushes to synthesize glyphs is that they are flexible and can be easily stretched to fit different skeletons. From each extracted trunk segment, we reconstruct a brush by estimating a pair of thickness functions $(C_1(t), C_2(t))$ (Fig.5) that fit the thickness values along each side of a skeletal path. In the learning step, this brush representation provides a convenient way to extract features, such as thickness and curvature.

4.2. Learning and Inference

The objective of the learning step is to model the part similarities, which imply typographic styles across typefaces. As

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

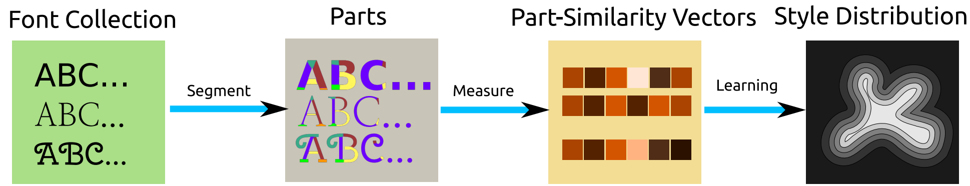

Figure 6: *Overview of our learning stage.*

representing typographic styles with fixed sets of rules is insufficient, we therefore seek for data-driven methods that allow style mixing and inference. In this section, we first introduce our part-similarity vector (PSV), which implicitly encodes the rules for reconstructing a complete typeface from its parts. The parts are compared using appropriate similarity and distance metrics, and collected into the PSVs. We then learn a distribution over PSVs using Bayesian Gaussian Process Latent Variable Model (BGPLVM). This distribution can be finally used to infer a compete PSV (a novel set of styles) from the partial PSV calculated from the user's input glyphs. Fig. 6 gives an overview of this step.

Intermediate Feature Vectors and Distance Metrics. Before we can proceed to extract the PSVs, we need to introduce a set of intermediate features, which are used for comparing brushes, skeletons and caps. The similarities between these elements are used to establish the PSVs and to quantitatively evaluate our model (Sec. 6). For brushes, we sample each of the parametric curves C_1, C_2 at $\{t_i\}$ to get the curvature and thickness pairs $\{(c_i, d_i)\}$, where c_i is the curvature and d_i is the thickness. Similarly, we use the curvatures $(\{c_i\})$ of the trajectory paired with the angular differences (denoted as $\{d_i\}$ as well for simplicity) between the tangents at a point on the trajectory and the horizontal axis to represent skeletons. We then measure the similarities between these pairs by using a weighted Euclidean distance metric,

$$d = \sum_i (w_c|c_i - c_i'|^2 + w_d|d_i - d_i'|^2)^{1/2},$$

where w_c, w_d are the weights controlling the contributions for curvature versus thickness (for brushes) or curvature versus angle (for skeletons). For stroke outlines, we used $w_d = 1$ for thickness and $w_c = 0.1$ for curvature. For skeletons, we used $w_d = 1$ for angle and $w_c = 0.2$ for curvature. It is worth mentioning that the curvature values were mapped to the range $[-1, 1]$ prior to the similarity measurement. For caps, their similarities are computed directly on the outline points using a modified Hausdorff distance (MHD) [DJ94] and then also normalized. Specifically, the MHD is defined as follow:

$$\max(d(X,Y), d(Y,X)),$$

where $d(X,Y) = \frac{1}{|X|}\sum_{x_i \in X} d(x_i, Y)$ with $d(x_i, Y) = \min_{y_j \in Y} \|x_i - y_j\|$, and x_i and y_j are the locations of the sampled outline points.

Part-Similarity Vector. A part-similarity vector (PSV) comprises all pairwise similarities between *all the parts of all the glyphs* in a typeface. Consider a typeface consisting of N glyphs where each glyph has n_i parts. The total number of parts is $N_p = \sum_{i=1}^{N} n_i$. The PSV is defined as

$$\left[d(p_0,p_1),\ldots,d(p_i,p_j),\ldots,d(p_{N_p-1},p_{N_p})\right],$$

where (p_i, p_j) is a unique unordered pair of glyph parts and $d(\cdot)$ is a distance. The details about the feature vectors and the distances are mentioned previously. The total length of a PSV is $N_p * (N_p - 1)/2$. We calculate a separate PSV for brush, cap, and skeleton, and hence the style of each font is represented by a set of 3 PSVs. Each element in the PSV is considered as a possible "copying rule" from p_i to p_j and vice versa. When the parts' orientation (e.g. the order of a brush's thickness values) is taken into account, we also add the distances between the flipped versions of the parts into the feature vector.

BGPLVM. In our work, the PSV distributions are learned using Bayesian Gaussian Process Latent Variable Model (BGPLVM), a method that has shown great performance in manifold learning as well as data completion. A mixture density model with latent variables (e.g., BGPLVM) assumes that the data is generated from multiple components, with each component explaining a part of the observations. This suits our purpose well, since we want to model different design styles hidden in each typeface. Moreover, we prefer a generative model over discriminative ones since we would like to draw a set of novel styles for synthesization.

BGPLVM [TL10] is a variant of Gaussian Process Latent Variable Model (GPLVM) [Law04] that uses variational methods for model learning and inference. BGPLVM and GPLVM are dimensionality reduction methods with several advantages, including the ability to model nonlinear mappings from latent to observed space using kernels, and robustness on small datasets with large dimensionality due to its usage of Gaussian Processes. The kernel used in our implementation is the Radial Basis Function (RBF) kernel. To perform style prediction with BGPLVM, an incomplete PSV is first calculated from the input glyphs by comparing the parts extracted from them (Section 4.1). Next, we search the low-dimensional latent space induced by BGPLVM for the latent point that yields the highest likelihood of the incomplete PSV. To speed up the search process for the optimal

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

latent point, we initialize the search using the latent point of the most similar complete PSV from the dataset. Finally, the optimal latent point is projected back to the PSV space to obtain the full PSV. The above steps are performed independently for brush, cap and skeleton.

4.3. Synthesization

After the inference step, we obtain 3 complete PSVs (for brush, cap and skeleton) which tell how much a part is similar to another in the final typeface. The transferring rule for a part is extracted from these PSVs by picking the optimal vector element that is related to this part. Formally, to reconstruct part p_i, we select an element that minimizes:

$$\underset{j}{\operatorname{argmin}}\, x_{i,j},$$

where $x_{i,j}$ is an element of a PSV, which measures the similarity between parts p_i and p_j. To reconstruct a glyph, we first need to obtain the transferring rules for skeleton, brush and cap separately, and then transfer the parts accordingly. The details of the transferring algorithms are given below.

Skeleton Retrieval and Deformation. Skeletons tend to be universal across typefaces for individual glyphs (as the skeleton represents the semantic meaning of the glyph). Thus we could simply retrieve from the database a set of skeletons similar to the skeletons of the input glyphs. The score function for skeleton retrieval is defined as follows:

$$\underset{f \in F}{\operatorname{argmin}} \sum_{i=1}^{N} \min d(g_i, g_i^f),$$

where F is a set of fonts in the database, N is the number of input glyphs, g_i is the feature vector (concatenated locations of outline vertices) of the i-th input glyph, and g_i^f is the feature vector of the corresponding glyph in the database. The minimum distance d is taken over rigid transformations of the glyph, and in our implementation we use a standard Iterative Closest Point (ICP) method [RL01] with $d(\cdot)$ as the sum of distances of corresponding points found by ICP (Fig. 7 (bottom)). Since the retrieved skeletons do not always match well with the input skeletons, we take a further step and deform individual parts of the template skeletons. Given the transferring rules extracted from the PSV for skeleton, we first calculate the vectors between the corresponding points on the template part and the input part, and then add the same vectors to a relevant template part according to the rules. The deformed skeleton part is then smoothed by fitting a smooth curve over it if necessary.

Stroke Body Synthesization. As we represent a brush as a pair of thickness functions $(C_1(t), C_2(t))$, to synthesize a stroke body given a skeleton $S(t)$ we reconstruct the outline by sampling thickness values from $C_1(t)$ and $C_2(t)$. The coordinates of a pair of points on the outline at $t = t_i$ are calculated as $\mathbf{p}_{t_i} = n(S(t_i)) \times C_1(t_i) + S(t_i)$ and $\mathbf{p}'_{t_i} = -n(S(t_i)) \times C_2(t_i) + S(t_i)$, where $n(S(t_i))$ is the unit normal vector at $S(t_i)$.

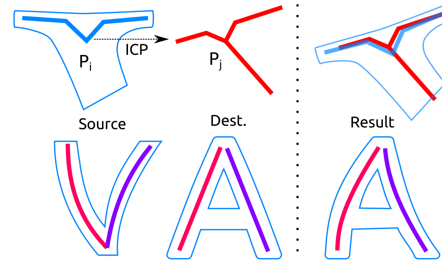

Figure 7: *ICP fitting is used to transform the given cap or skeleton to the reference one.*

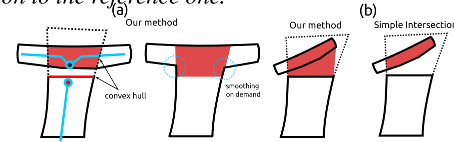

Figure 8: *Our part merging method. (a) simple case, where we extend the body and take the intersection with the cap. (b) a case when simple extension (dotted black lines) and intersection do not work. We use convex hull to fill in the gap.*

Cap Synthesization. To synthesize the cap, we transform the cap part outline to match the corresponding skeletal part of the new glyph. To do this, as illustrated in Fig. 7 (top), we rely on ICP to match path P_i from the source skeleton to path P_j in the destination skeleton. We then transform the source outline with the transformation matrix **A** returned by the ICP algorithm. Although this method seems naive, it can produce very reasonable results. Another solution might be to encode the direction of transferring into the PSV as well by adding the similarities of the flipped versions of a part. In our system, this approach is implemented.

Merging and Smoothing Parts. To smoothly merge the parts together, we first find the joints between parts. The terminals of a part are assigned "head" or "tail" labels following the predefined rules such as left-to-right and top-to-down. We also use "mid" to refer to a point in between "head" and "tail". Given 2 parts, we define 5 types of joints: *head-head* (HH), *head-tail* (HT), *tail-tail* (TT), *head-mid* (HM) and *tail-mid* (TM). For {HH, TT, HT} we extend the parts along the tangent vector at the ending vertices and take the intersection. Note that only joints between two stroke bodies can have HH and TT labels. For HM and TM cases, we only extend the head/tail part before taking the intersection. While taking a simple intersection between extended parts works in easy cases, it can fail in difficult cases like (b) in Fig. 8, where parts' sizes are too different. We address this by finding the intersection first, and then calculating the convex hull of the points on the contour of the intersection *and* the points near the merging areas (red line in the first figure). Additional local smoothing operations can be carried out afterwards to remove small artifacts near the joints.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Char	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M
All	4	9	9	6	5	6	4	3	2	4	6	3	4	7	5	7	3	7	5	7	7	3	5	5	7	9
Body	3	4	4	3	2	3	2	1	2	2	3	1	2	3	2	3	1	3	2	3	3	1	2	3	3	4
Char	q	w	e	r	t	y	u	i	o	p	a	s	d	f	g	h	j	k	l	z	x	c	v	b	n	m
All	4	9	4	5	4	6	5	4	2	4	4	3	4	4	5	5	4	7	3	7	7	3	5	4	5	7
Body	2	4	3	2	2	3	2	1	2	2	2	1	2	2	4	2	1	3	1	3	3	1	2	2	2	3

Table 1: *Number of parts for each glyph. "All" and "Body" are the numbers of parts (bodies plus caps) and bodies only, respectively.*

5. Qualitative Results

In the experiments, we focused on upper-case characters of Latin characters. Results on lower-case characters were also shown for reference. The number of parts for each character is summarized in Table 1. For special cases like "O" and "Q", we vertically split the main circles into two halves. Our dataset contains 88 training fonts for upper case and 57 training fonts for lower case selected from a public font dataset [Goo]. We provided both qualitative (this section) and quantitative evaluations (Sec. 6).

We evaluated our method qualitatively with 3 experiments: typeface reconstruction (Sec. 5.2), novel typeface synthesization (Sec. 5.3) and edit propagation (Sec. 5.4). In the first test, we visually examined how well our method can reconstruct an existing font. The results are shown along with ground truth typeface for comparison. In the second test, we synthesized novel typefaces from user-designed input glyphs. In the third test, we demonstrated the editing capability of our system. To compare with the method of [SI10], we show our outputs given the input taken from the "failed case" in their paper. Additionally, we demonstrate how novel transferring rules can be drawn from the part-similarity distribution and how a fixed rule did not fit to all cases.

5.1. Learning versus Retrieval

We compared our novel transferring rules produced by BGPLVM with two methods that simply retrieve the rules from the most similar fonts. One retrieval based method used our PSV feature while the other used intermediate features detailed in Section 4.2 for retrieving similar transferring rules. The Euclidean distance was used for retrieval. Fig. 9 shows differences in transferring rules produced by 3 different methods, namely BGPLVM with PSV, retrieval with PSV, and retrieval with intermediate features.The usual thick-and-thin style was slightly modified in the given glyphs {F, M, O}. The pink and yellow parts of 'F' were thicker than the purple part. Similarly, the thickness of parts in 'M' was changed. Retrieval with intermediate features tended to produce rather "random" rules of transferring. This is because it only matches the appearances of fonts. In this case, a font with single brush type was obtained for reference, resulting in vague transferring rules. In contrast, the rules inferred by BGPLVM exhibited mixtures of different styles. The style of 'F' was successfully transferred to 'E', 'H', and 'K', while the style of 'M' appeared in 'N' and 'W'. The rules retrieved with PSV gave fairly reasonable results as they rely on part-similarity instead of glyph appearance.

It is interesting to see if a fixed set of rules drawn directly from a training sample would work on different inputs. Fig. 10 shows such results of using a fixed set of rules, extracted from a reference font. The given glyphs were {H,C} for the top figure, {F,M,O} for bottom. Fig. 10 (top) shows variations of the reference font with different inputs. Notice how a single set of rules produced very different looking typefaces. Fig. 10 (bottom) shows two comparisons of fixed rules versus predicted rules. In the comparison, the bottom part of 'M' was slightly deviated from the standard thick-and-thin rule. This change was not reflected in 'N' and 'U' for the case of fixed rules as these glyphs are out of rhythm with 'M'. More seriously, since the reference font only has straight strokes, the orientations of glyphs were completely ignored. Thus, the results with fixed rules had rather random stroke orientations. In contrast, the learning method produced correct stroke orientations and nicely transferred the style of 'M' to 'N' and 'U'. Similarly, in the second comparison, with the fixed rules the trunks of {B, D, H, I, J, K, L, R, P} were all transferred from the right-most part of 'M', which made them look inharmonious to the given glyphs. There was no such problem with our learning method.

5.2. Reconstruction with Ground-truth Skeleton

We show the results of reconstructing fonts from 3 sample glyphs taken from the original fonts. The skeletons were both taken from the original fonts and the retrieved fonts. The purpose is to visually test whether our method can correctly recover the styles associated with a font given only a few example glpyhs. The two fonts in the experiment are "Adamina" and "CherrySwash". In Fig. 11, rows 1 and 7 highlighted in orange color are ground-truth fonts. Rows 2, 3 and 8 are the reconstruction results with ground-truth skeletons and synthesized outlines; glyphs in pink were those given to the synthesizer. The glyphs were chosen because they contain only a small set of strokes. From the results, we can see that most of the brush strokes were correctly aligned. For example, the curvy part of {P, R, D, C, G, Q} were transferred from 'O'. Because of the correctly predicted transferring rules, those characters looked almost identical to the ground-truth. 'S' is a special case: while it does not look similar to 'O' or 'C' in shape, the underlying brushes are actually the same. Moreover, the rules of using thick and thin strokes were predicted quite nicely. For example, {A, B, E, F, H, K, L, M, N, W, X, Y} are the characters that have complex composition rules and were correctly reconstructed from {I, C, V}. The only failed case in row 3 was 'Z'. However, 'Z' was correct in the case of input {H, V, O} (row 2). This suggests an interesting feature of our system: given the newly reconstructed glyphs, users can specify which glyphs

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Figure 9: *Visual comparison of three transferring rules obtained by (A) BGPLVM with PSV, (B) retrieval with PSV, and (C) retrieval with intermediate features. Input glyphs are highlighted with red boxes. Parts bearing the same colors share the same brushes.*

Figure 10: *Different outputs and inputs given a fixed set of rules extracted from a reference font "Amiri-Regular". (Top) variations given different glyphs of "C" and "H" as inputs. (Bottom) fixed rules versus predicted rules. The input glyphs were {F, M, O} for both cases. Parts in the same colors were reconstructed with the same brushes or caps.*

are good by simply clicking on them. The "good" glyphs are then put back to the system to re-synthesize the "bad" ones, which will be demonstrated in Sec. 5.3. In rows 2 and 3, 'U' and 'J' are failure cases to some extent. Since we did not provide the system with any similar strokes (the closest one is the left part of 'O'), it is impossible to recover the perfect glyph from these examples. However, since these transferring rules usually have low probabilities, we can show to the designer which parts might need their correction. Fig. 12 demonstrates this idea where the confidences of stroke transfers are visualized with a color map.

The caps were also synthesized equally well. Notice that the top caps from 'T' were correctly transferred to {S, Z, E, F, G} (row 3). One may argue that these rules are fixed and are available in design books so we do not need learning to predict. In fact, the result in Line 7 suggests differently. In "CherrySwash", a different style compared to row 2 is observed. The tip of 'A' is no longer similar to the bottom of 'V'. The caps of 'G' and 'S' are almost invisible while the left caps of {H, K, M, N} are not the same as the right ones as in the "Adamina" case. Our inference engine managed to correctly predict these particular rules with few mistakes ('Z' and 'X').

5.3. Reconstruction without Ground-truth Skeleton

Next we test whether a typeface can be reconstructed without ground-truth skeletons. Given the input glyphs, we retrieved the skeletons using the method described in Section 4.3. We only performed skeleton transfer when the similarity between two skeleton parts was greater than a threshold. In Fig. 11, rows 5 and 10 show the synthesization results with skeletons retrieved from fonts in rows 6 and 11. For the case of "CherrySwash", the results look quite similar to the ground-truth, despite the synthesized ones being slightly thicker. This is because the sizes of the reference skeletons were smaller than the real ones. In practice, a user could manually scale up the reference skeletons, thus reducing the thickness of the synthesized glyphs. In row 10, although the reference font (in row 11) looks completely different from the ground-truth one (in row 7), our method still worked relatively well. Note that the skeletons of 'C' and 'G' were automatically deformed according to the examples. More results for skeleton transfer can be found in Fig. 15.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Figure 11: *Typeface reconstruction results for fonts "Adamina" and "CherrySwash". Glyphs in pink are the inputs. (1) and (7) are the ground-truth fonts. (2), (3) and (8) are synthesized fonts with ground-truth skeletons from (1) and (7), respectively. (4) and (9) are customized fonts after edit propagation. (5) and (10) are synthesized fonts with retrieved skeletons, from fonts (6) and (11), respectively.*

Figure 12: *Transferring probabilities are visualized with color mapping. The input glyphs were "C" and "H".*

In Fig. 13 we show our synthesization results given a single example. The glyph 'A' was taken from a failed case of the method in [SI10]. The first line shows initial synthesization results, which were not very good except for the glyphs {H, O, K, N, U, V, W, X}. As mentioned in the last subsection, it is possible to put these "good" examples back to the system to improve the "bad" ones. In this case, the user identified 'I' as a good example. In row 2, we can see significant improvements over the initial results as the brush orientation was corrected (small to big in top-down direction). Row 3 was the result of letting a user correct the locations of the caps of the 'F' glyph, resulting in the changes in {C, E, J, S, T, Z}. This feature did not exist in [SI10] since the appearance of synthesized glyphs was totally dependant on the input template glyph in their method.

5.4. Edit Propagation and Use Cases

One advantage of our approach is its editing capability. While other systems provide very limited ways to customize a font, editing fonts in our system is very easy and intuitive. Rows 4 and 9 of Fig. 11 show the results of edit propagation. In Fig. 11, only 4 parts were modified to result in row 3. The novel glyphs appeared to carry well the spirit of the input glyphs given by the designer. In Fig. 14 we also show 5 sample designs with the glyphs taken directly from the synthesization results. One interesting feature of our approach is that it adapts to changes in the inputs to produce the best possible results. Fig. 15 demonstrates this capability. Since the entire system works on vectors, one can expect that the novel glyphs are in high-resolution and ready to use.

5.5. Lower Case

Although we did not show many results for lower-case characters, the framework works equally well on them. Figure 16 shows the synthesization results for the lower case.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Figure 13: *Font reconstruction from a single input glyph. First, 'A' (Red) was given to the system.(1) Initial results. (2) and (3) Results were refined by letting the user choose/change the desired glyphs (highlighted in orange). The glyphs affected by the change in the upper row are shown in Jade blue. (4) The final results after 3 rounds of refinement. (5) Results from [SI10] given the same input as the case in (1).*

Figure 14: *Some sample designs that used our generated glyphs. "PACIFIC" was designed with the glyphs in Fig. 10. "GRAPHICS" and "BEST CORN" are from rows 4 and 8 in Fig. 11. "FLY HIGH ABOVE" is from 'H' and 'F' in the same text. "SHANGHAI" is from Fig. 13.*

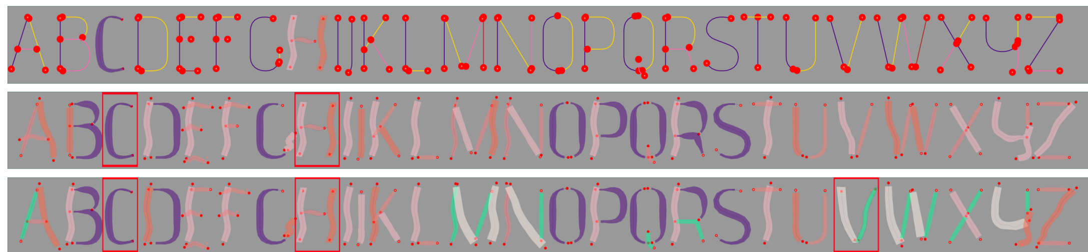

Figure 15: *Different inputs given different transferring rules. (1st row) the template skeletons along with 2 input glyphs (red boxes). (2nd row) synthesization results with 'C' and 'H'. (3rd row) synthesization results with 'C', 'H' and 'V'. Parts bear the same colors as input parts are transferred from them.*

Figure 16: *Lower case typeface reconstruction and edit propagation for font "Alegreya-Regular". (1st row) ground truth, (2nd row) reconstruction result using retrieved skeleton, (3rd row) customized version where strokes were made thicker.*

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Results in accuracy scores			
	Brush	Cap	Skeleton
Truth	1698*	1862*	1718*
GPLVM-PSV	1591*	1471*	1668
Retrieval-PSV	1557	1350	1656
Retrieval	1545	1375	1662

Table 2: *Quantitative results. Accuracies measured by total similarities between reconstructed parts and reference parts (higher is better). * indicates that the result in a cell was significantly better than the cell below (t-test, p-value < 0.0001).*

6. Quantitative Results

To further justify our learning approach, we also conducted quantitative experiments to measure the performances of BGPLVM and two other retrieval approaches by comparing the total similarities between the reconstructed parts and the reference parts, where the latter was chosen according to some transferring rules, either retrieved or inferred. The retrieval methods are mentioned in Sec. 5.1. We also show the best possible reconstructions by extracting the rules from the ground-truth part similarities. Each method was repeated 50 times on random training-testing splits with 60% used for training and 40% used for testing. The initial glyphs were chosen by taking the top scoring characters from the ground-truth results. Some best scoring character triplets were {E, O, M}, {E, O, W}, {E, Q, M}, {E, B, W}. The same sets of initial glyphs were given to each method in each test. The average results for each method and pairwise t-test results were reported below.

Table 2 shows that the learning approach was significantly better than the retrieval-based approaches for the cases of *Cap* and *Brush* predictions. The scores in the table were calculated by summing over the ground-truth similarities between the reconstructed parts and reference parts. Higher values mean better results. The "Truth" results were the best possible reconstruction scores given the inputs (i.e., the best possible transferring rules, according to the ground-truth similarities). The performance of BGPLVM was close to the best solutions for the *Brush* and *Skeleton* cases. It is interesting to see that we did not gain much when learning on the *Skeleton* data. It is because glyph skeletons appeared to be universal among typefaces. Thus, it was sufficient to use the template skeletons and deform them to meet the given glyphs as shown in Sec. 4.3.

7. User Interface

We designed a simple user interface that allows users to interact with our font synthesization engine. The interface has been integrated into an open source vector drawing software called *Inkscape* [Ink]. Fig. 17 shows how a user can interact with our interface to create novel fonts from scratch. Please see the accompanied video for more information.

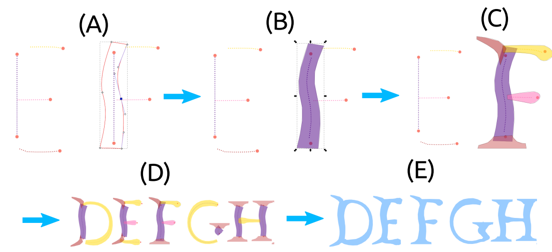

Figure 17: *User interface for interactive font design. (A) A user inputs raw strokes by drawing over template skeletons. (B) The input path is snapped to a skeletal part. (C) Finished inputting parts for a single glyph. (D) Synthesized glyphs. (E) Final merged and smoothed typeface. The whole process took about 3 minutes for a user with experience in vector drawing.*

8. Conclusions and Limitations

In this paper we presented a novel framework for font synthesization and customization using a part-assembly approach. The framework consists of 3 main components: (1) stroke-based glyph representation, (2) probabilistic style learning and inference engine, and (3) typeface synthesization. We demonstrated the performance of our system on different tests such as font reconstruction, novel font synthesization and edit propagation. Both quantitative and qualitative results appeared to be promising.

However, our approach still suffers from a number of limitations. First, since the skeleton deformation does not take into account neighboring parts, smooth alignments might not be always obtained. Fig. 18 shows two cases when the skeletons are not well aligned and might require user intervention. In practice, manual alignments are not often needed for stroke body since we can always find a skeleton from the database that closely matches the input glyph, thus largely reducing the displacements. It is more common for users to align the caps of characters like {W,N,Z}, since the presence or absence of the caps in these characters varies from font to font. Second, cap alignment depends on ICP matching, which does not always produce desired results when reference skeletons are too different from the input. A possible solution is to encode the orientations of the caps into the PSV, as mentioned in Sec. 4.2. Another promising future work is to extend our part similarity approach to other geometric objects like decorative patterns.

Acknowledgements

We would like to thank the anonymous reviewers for their help in improving the paper and the authors of GPy library for the BGPLM code. This work was partially supported by grants from the Research Grants Council of HKSAR, China (Project No. CityU 113513, CityU 11204014, CityU 123212).

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Figure 18: *Failed cases when parts are not perfectly aligned.*

References

[AAAG96] AICHHOLZER O., AURENHAMMER F., ALBERTS D., GÃĎRTNER B.: A novel type of skeleton for polygons. In *J.UCS The Journal of Universal Computer Science*, Maurer H., Calude C., Salomaa A., (Eds.). Springer Berlin Heidelberg, 1996, pp. 752–761. 4

[Bos12] BOSLER D.: *Mastering Type: The Essential Guide to Typography for Print and Web Design.* F+W Media, 2012. 1

[CK14] CAMPBELL N. D. F., KAUTZ J.: Learning a manifold of fonts. *ACM Trans. Graph. 33*, 4 (July 2014), 91:1–91:11. 1, 2, 3

[DJ94] DUBUISSON M.-P., JAIN A. K.: A modified Hausdorff distance for object matching. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision and Image Processing., Proceedings of the 12th IAPR International Conference on* (1994), vol. 1, IEEE, pp. 566–568. 5

[Goo] GOOGLE I.: Google WebFonts. https://www.google.com/fonts. 7

[HH01] HU C., HERSCH R.-D.: Parameterizable fonts based on shape components. *Computer Graphics and Applications, IEEE 21*, 3 (May 2001), 70–85. 3

[Hof85] HOFSTADTER D. R.: *Metamagical Themas: Questing for the Essence of Mind and Pattern.* Basic Books, Inc., New York, NY, USA, 1985. 3

[Ink] INKSCAPE: Inkscape. https://inkscape.org/. 11

[JPF06] JAKUBIAK E. J., PERRY R. N., FRISKEN S. F.: An improved representation for stroke-based fonts. In *Proceedings of ACM SIGGRAPH* (2006). 4

[KCKK12] KALOGERAKIS E., CHAUDHURI S., KOLLER D., KOLTUN V.: A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 55. 3

[KLM*13] KIM V. G., LI W., MITRA N. J., CHAUDHURI S., DIVERDI S., FUNKHOUSER T.: Learning part-based templates from large collections of 3d shapes. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 70. 3

[Knu79] KNUTH D. E.: *TEX and METAFONT: New Directions in Typesetting.* American Mathematical Society, Boston, MA, USA, 1979. 2, 3

[Kru64] KRUSKAL J. B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika 29*, 1 (1964), 1–27.

[Lau09] LAU V. M. K.: Learning by example for parametric font design. SIGGRAPH ASIA '09. 3

[Law04] LAWRENCE N. D.: Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems 16* (2004), 329–336. 5

[LYFD12] LU J., YU F., FINKELSTEIN A., DIVERDI S.: Helpinghand: Example-based stroke stylization. *ACM Trans. Graph. 31*, 4 (July 2012), 46:1–46:10. 3

[MHS*14] MA C., HUANG H., SHEFFER A., KALOGERAKIS E., WANG R.: Analogy-driven 3d style transfer. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 175–184. 4

[OLAH14] O'DONOVAN P., LĪBEKS J., AGARWALA A., HERTZMANN A.: Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 92. 3

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on* (2001), IEEE, pp. 145–152. 6

[SI10] SUVEERANONT R., IGARASHI T.: Example-based automatic font generation. In *Proceedings of the 10th International Conference on Smart Graphics* (2010), SG'10, pp. 127–138. 1, 2, 3, 7, 9, 10

[SR98] SHAMIR A., RAPPOPORT A.: Feature-based design of fonts using constraints. In *Electronic Publishing, Artistic Imaging, and Digital Typography* (1998), Hersch R., AndrÃľ J., Brown H., (Eds.), vol. 1375 of *Lecture Notes in Computer Science*. 2

[TL10] TITSIAS M., LAWRENCE N.: Bayesian Gaussian process latent variable model. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2010). 5

[XLCP05] XU S., LAU F., CHEUNG W. K., PAN Y.: Automatic generation of artistic chinese calligraphy. *Intelligent Systems, IEEE 20*, 3 (2005), 32–39. 3, 4

[Zit13] ZITNICK C. L.: Handwriting beautification using token means. *ACM Trans. Graph. 32*, 4 (July 2013), 53:1–53:8. 3

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.