Generating Font Variations Using Latent Space Trajectory

Sotaro Kanazawa
kanazawa-sotaro317@g.ecc.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

I-Chao Shen
jdilyshen@gmail.com
The University of Tokyo
Tokyo, Japan

Yuki Tatsukawa
tatsukawa-yuki537@g.ecc.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

Takeo Igarashi
takeo.igarashi@gmail.com
The University of Tokyo
Tokyo, Japan

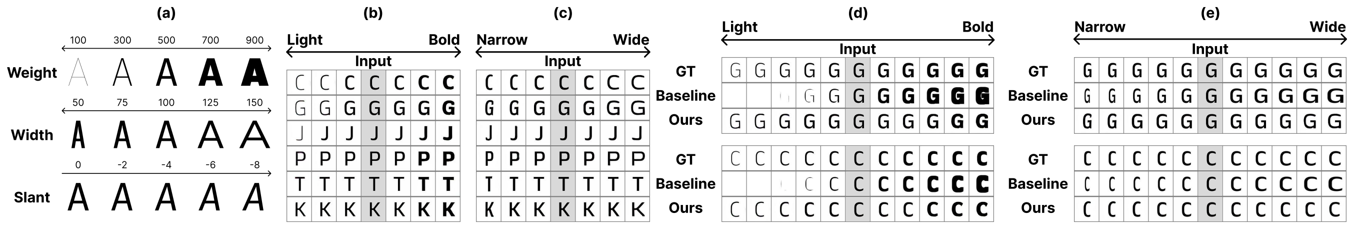

Figure 1: (a) Examples of variable fonts. There are several axes of font properties, such as weight, width, and slant. (b) Font variations for weight axis generated by our method. (c) Font variations for width axis generated by our method. (d) (e) Comparison between our method (Ours) and the baseline methods (Baseline) (d) for weight axis (The baseline method is morphological transformation) (e) for width axis (The baseline method is horizontal scaling). Both for (d) and (e), GT refers to bitmap font variations from an existing variable font.

ACM Reference Format:
Sotaro Kanazawa, I-Chao Shen, Yuki Tatsukawa, and Takeo Igarashi. 2024. Generating Font Variations Using Latent Space Trajectory. In *SIGGRAPH Asia 2024 Posters (SA Posters '24), December 03-06, 2024.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3681756.3697933

1 INTRODUCTION

Fonts are essential for conveying information with character, and various kinds of fonts exist around you. Sometimes, you change font properties to achieve more effective communication using them. You can change fonts into bold and italic ones in tools such as PowerPoint. However, you can only change the parameters of the properties into fixed ones. Fonts that have properties with fixed parameters in this way are called static fonts. It is desirable to change the parameters continuously to enable a more flexible design. To solve this problem, fonts that allow you to freely change the parameters of font properties, called variable fonts, have rapidly spread in recent years (Figure 1 (a)). These fonts are available in various design tools, such as Adobe Illustrator.

However, professional typeface designers currently design variable fonts, and the design process is too detailed and takes too much time [Scheichelbauer 2024]. Specifically, they first have to design master fonts at both ends (For example, when they design variable fonts for the weight axis, they have to create the lightest and boldest fonts) in every detail. Then, they interpolate these two master fonts. The problem is that they have to repeat this process for each character and each attribute. Therefore, the entire process becomes labor-intensive and time-consuming.

In this way, the design process of variable fonts is a heavy burden on typeface designers. At the same time, the design process of static fonts is also too detailed and time-consuming. Therefore, these days, research on font generative models, generative models that can automatically generate static fonts, is becoming more and more popular [Jahanian et al. 2020]. In this study, to assist in designing variable fonts, we extend these font generative models to the automatic generation of variable fonts by utilizing the latent space trajectory of font generative models.

We demonstrated that our method can generate high-quality bitmap font variations for multiple axes (Figure 1 (b) (c)).

2 METHOD

Our proposed method is to generate bitmap font variations from the bitmap of a single-character static font. We pre-train a font generative model with Roman static fonts. Our method utilizes DG-Font [Xie et al. 2021] as the font generative model. We provide the bitmap of a single-character static font as input to both style and content encoders of DG-Font. First, we optimize the direction of a trajectory in the latent space of DG-Font according to the desired attribute. After that, we map the input bitmap of the static font to the latent space with the encoder of DG-Font. Finally, we move the mapped latent vector on the trajectory corresponding to the desired attribute and generate bitmap font variations.

Regarding the optimization of a trajectory in the latent space, we leverage the method similar to [Jahanian et al. 2020]. Before the

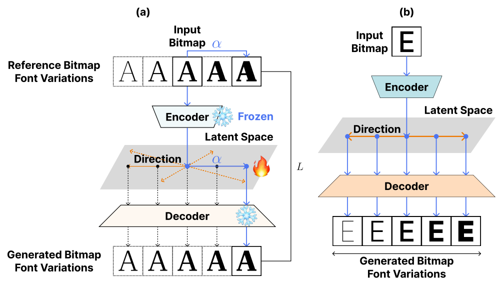

Figure 2: (a) The process of optimizing the direction of a latent space trajectory. We first prepare rasterized variable fonts as reference font variations. Then, we move the parameter of reference font variations to a certain degree α. At the same time, we move the latent vector mapped from the middle instance of reference font variations to the same degree. After that, we obtain the bitmap from the moved latent vector and calculate the loss with the ground truth. (b) The process of generating bitmap font variations. After obtaining the trajectory optimized for the desired property, we generate bitmap font variations from the bitmap of a single-character static font by leveraging the direction of this trajectory.

optimization process, we first prepare bitmap font variations as references by rasterizing real-world variable fonts with an axis of the desired property. Then, in the optimization process, we update the parameter of the trajectory direction to follow the desired property utilizing the reference font variations.

We also explain the concrete workflow of the optimization (Figure 2). We use α as the parameter representing the degree of transforming input static font bitmap. We move the parameter of variable font to the same degree as α, and obtain the reference font bitmap $\mathsf{Var}(I, \alpha)$ with respect to reference bitmap I. We define Enc as DG-Font encoder and Dec as DG-Font decoder. We use the bitmap rasterized from the training variable font with the middle parameter as the input in the optimization process. First, we obtain $\mathsf{Enc}(I)$ from the input bitmap I with the encoder. Then, we obtain $\mathsf{Enc}(\mathrm{I}) + \alpha w$ by moving the latent vector to the degree of α along the latent vector w that represents the direction of the trajectory in the latent space. Finally, we obtain font bitmap $\mathsf{Dec}(\mathsf{Enc}(I) + \alpha w)$ from the moved latent vector with the decoder. Regarding loss function, we define the L2 loss below:

$$L2(\mathsf{Dec}(\mathsf{Enc}(I) + \alpha w), \mathsf{Var}(I, \alpha))$$

Then, we optimize the direction of the trajectory to minimize the following objective function:

$$w^* = \arg\min_{w} \mathbb{E}_{I,\alpha}[L2(\mathsf{Dec}(\mathsf{Enc}(I) + \alpha w), \mathsf{Var}(I, \alpha))]$$

3 EXPERIMENT

We generated bitmap variations of the Roman fonts for two attributes (weight and width) and evaluated our method. We compared the geometric transformation (baseline methods) and our method quantitatively and qualitatively regarding these two attributes. In the quantitative evaluation, we compared them in terms of generalization ability to unseen fonts and characters.

As the baseline method, we used geometric transformation (Figure 1 (d) (e)). Specifically, we used morphological transformation (a geometric transformation that erodes or dilates an object in an image) for weight axis and horizontal scaling for width axis.

3.1 Experimental Setup

We collected 250 variable fonts for weight axis and 60 for width axis, respectively, and divided them into training data (variable fonts for optimization) and test data (variable fonts for evaluation). In evaluating the generalization ability to unseen fonts, we used 62 (0-9, a-z, and A-Z) characters and divided fonts into (train, test) = (200, 50) for weight axis, and (train, test) = (50, 10) for width axis. In evaluating the generalization ability to unseen characters, we used (weight, width) = (250, 60) fonts and divided characters into 56 (0-9, a-z, and A-T) training characters and 6 (U-Z) test characters.

3.2 Results

Table 1 shows the comparison results for weight and width axes. As a result, our method is quantitatively superior to the baseline method (morphological transformation) for weight axis. On the other hand, for width axis, the baseline method (horizontal scaling) is superior to our method.

	Ours (L2 ↓)	Baseline (L2 ↓)
unseen font (weight)	**0.1695**	0.2590
unseen character (weight)	**0.1429**	0.1908
unseen font (width)	0.2418	**0.09437**
unseen character (width)	0.2338	**0.07793**

Table 1: Comparison with geometric transformation (The baseline methods).

Figure 1 (d) (e) refer to the comparison of generated font variations by geometric transformation (Baseline) and our methods (Ours) with bitmap font variations obtained from ground truth variable fonts (GT).

(d) is the comparison for weight axis. In particular, font variations generated by the baseline method disappear as the font becomes thinner. On the other hand, our method can maintain its form.

(e) is the comparison for width axis. Regarding the weight of fonts, ground truth variable fonts have a consistent weight independent of the width. On the other hand, The weight of font variations generated by the baseline method depends on their width. At the same time, our method can preserve the weight of generated font variations regardless of their width. Therefore, the actual quality of our method is higher than that of the baseline method, even though the baseline method is superior to our method in the quantitative evaluation.

ACKNOWLEDGMENTS

This work was partially supported by JST AdCORP, Grant Number JPMJKB2302, JSPS Grant-in-Aid JP23K16921, Japan, and a collaboration with Dentsu Digital.

REFERENCES

Ali Jahanian, Lucy Chai, and Phillip Isola. 2020. On the "steerability" of generative adversarial networks. In *International Conference on Learning Representations*.

Rainer Erich Scheichelbauer. 2024. Creating a variable font. https://glyphsapp.com/learn/creating-a-variable-font.

Yangchen Xie, Xinyuan Chen, Li Sun, and Yue Lu. 2021. DG-Font: Deformable Generative Networks for Unsupervised Font Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5130–5140.