DualVector: Unsupervised Vector Font Synthesis with Dual-Part Representation

Ying-Tian Liu[1] Zhifei Zhang[2] Yuan-Chen Guo[1] Matthew Fisher[2]
Zhaowen Wang[2] Song-Hai Zhang[1*]
[1] BNRist, Department of Computer Science and Technology, Tsinghua University [2] Adobe Research
{liuyingt20@mails., guoyc19@mails., shz@}tsinghua.edu.cn
{zzhang, matfishe, zhawang}@adobe.com

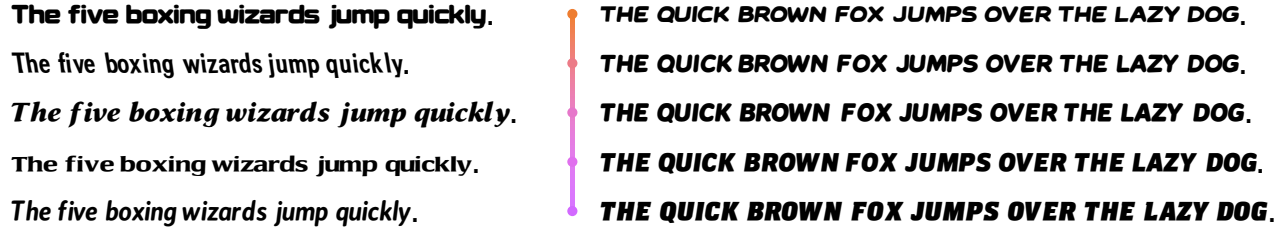

Figure 1. High-quality vector fonts (copyable on electronic devices) synthesized by DualVector (left), with smoothly-interpolated styles (right). Both sentences are pangrams. Please zoom in for details.

Abstract

Automatic generation of fonts can be an important aid to typeface design. Many current approaches regard glyphs as pixelated images, which present artifacts when scaling and inevitable quality losses after vectorization. On the other hand, existing vector font synthesis methods either fail to represent the shape concisely or require vector supervision during training. To push the quality of vector font synthesis to the next level, we propose a novel dual-part representation for vector glyphs, where each glyph is modeled as a collection of closed "positive" and "negative" path pairs. The glyph contour is then obtained by boolean operations on these paths. We first learn such a representation only from glyph images and devise a subsequent contour refinement step to align the contour with an image representation to further enhance details. Our method, named DualVector, outperforms state-of-the-art methods in vector font synthesis both quantitatively and qualitatively. Our synthesized vector fonts can be easily converted to common digital font formats like TrueType Font for practical use. The code is released at https://github.com/thuliu-yt16/dualvector.

1. Introduction

Fonts with various styles play an important role in content display and distribution. Excellent font design is time-consuming and labor-intensive. Recent machine learning innovations have made font generation possible, but how to automatically generate high-quality vector fonts remains a task of practical importance in the artistic and computer graphics and vision communities.

Benefiting from the development of image generation techniques, mainstream font synthesis methods [2, 12, 24, 41, 42] could generate pixelated glyph images. Despite the promising quality, images of glyphs incur aliasing artifacts on edges when discretely sampled, and thus are not competent for high-quality rendering or printing at arbitrary resolutions. To alleviate this problem, some methods [7, 34] adopt coordinate-based neural networks to model a glyph as a contiguous neural field, which have also shown great potential in modeling 3D geometry and scenes [7, 30, 32]. Although glyphs represented by the implicit field can be rendered at arbitrary resolutions, it is hard to preserve details in high-frequency regions such as edges and corners, not to mention the high computational costs as the network needs to be evaluated for every pixel. Researchers have made much effort to directly synthesize vector fonts [4, 27, 33, 39] in recent years, with the main difficulty lying in finding a representation of vector graphics that can be encoded or decoded effectively in a deep learning framework. One typical approach represents a vector shape as a sequence of drawing commands and adopts sequence modeling techniques such as recurrent networks and transformers. The drawbacks are twofold: (1) Modeling command sequences can be much harder than images. There are infinitely many command sequences that correspond to the same-looking shape, which brings ambiguities in learning and makes it hard to construct

*Corresponding author

an effective manifold for valid glyph shapes. (2) Ground-truth drawing commands are often required to provide sufficient supervision for high-quality modeling and synthesis.

To overcome these challenges, we first propose a dual-part vector font representation, where each glyph shape is the union of a fixed number of *dual parts*. Each *dual part* is formed by the subtraction of a "positive" and a "negative" geometric primitive. While there are many choices for the geometric primitives [6, 8, 25], we adopt closed Bézier paths for their great representational ability. They are also widely supported in digital font formats which makes it easy to convert our representation to these formats for practical use. We reduce the problem of predicting complicated drawing command sequences to predicting multiple basic primitives. From this perspective, both manifold learning and latent space interpolation become more feasible.

Based on the dual-part representation, we introduce DualVector, a method to learn such a representation for high-quality font modeling and synthesis from only glyph images without any vector supervision. A straightforward way to achieve this is to directly optimize the parameters of the Bézier curves with differentiable rendering techniques for vector graphics [22]. However, this approach easily gets stuck in the local minima as valuable gradients are only defined at shape intersections. Taking inspiration from implicit field training for 2D and 3D shapes [6, 25, 32], we supervise the occupancy value derived from the analytical expression of the Bézier curves and adopt an initialization strategy based on unsigned distance field (UDF) to provide dense gradients across the entire pixel space. For local detail fidelity, we also train a glyph image generation model and devise a subsequent contour refinement step to align the contour of the vector shape with that of the image by differentiable rendering [22]. We compare our approach with state-of-the-art methods in font modeling and generation and demonstrate the superior quality of our vector font outputs. Our main contributions are:

- A new dual-part font representation based on boolean operations of Bézier paths, which enables efficient shape modeling and unsupervised manifold learning.
- A method named DualVector that models both the dual-part and pixelated representation, and introduces a contour refinement step to obtain vector fonts with richer details as well as a UDF initialization strategy for better convergence.
- DualVector achieves state-of-the-art quality in font modeling and generation, with outputs that can be easily converted to common digital font formats.

2. Related Work

2.1. Glyph Image Generation

Benefiting from the development of image generation [10, 14, 16], either black-white [12, 15, 37, 40] or artistic glyph image generation [2,23,41,42] is well explored in the past decade. MC-GAN [2] synthesized ornamented glyphs for capital letters in an end-to-end manner from a small subset of the same style. Attr2Font [38] generated visually pleasing results according to style attributes and content references. Most of these methods follow the disentanglement of style and content codes, and so does the image generation branch in DualVector. But these glyph image generation efforts do not step outside the 2D pixelated representation which sets an upper bound for rendering quality, efficiency, and practicality.

2.2. Image Rasterization and Vectorization

Rasterization and vectorization are dual problems in image processing. Previous efforts on rasterization have typically focused on anti-aliasing [9, 11, 31] and high-performance rendering on modern graphics hardware [3, 17, 21, 26]. Traditional image vectorization approaches rely on region segmentation, line extraction, and contour tracing for gray-scale images [35], pixel arts [18], animations [44] and general images [1, 5, 19, 36, 45, 46]. Recently, researchers have turned to bridging the raster and vector domains [22, 29], enabling existing well-established algorithms for raster images to be also applied to vector representations. DiffVG [22] is a differentiable rendering technique for vector graphics enabling the optimization of vector primitive parameters based on raster criteria. In our work, it is employed to align the vector and pixel representation in the contour refinement step.

2.3. Vector Font Generation

SVG-VAE [27] was the first attempt to build a sequential generative model on SVG fonts. DeepSVG [4] developed a hierarchical transformer-based generative model for complex SVG icons generation and interpolation.Exploiting implicit neural representation, multi-implicits [34] modeled fonts with a permutation-invariant set of learned implicit functions which can preserve font features, such as smooth edges and sharp corners, and generate good interpolation results. Liu *et al*. [25] proposed a primitive-based representation, which views glyph shapes as the combination of primitives enclosed by implicit curves. Implicit representation-based methods can convert the zero-level set of their output fields to contour representation with the 2D marching cube [28] algorithm, although it produces multiple segmented lines and lacks accuracy and efficiency in rendering and editing. Im2Vec [33] can synthesize complex vector graphics with varying topologies without vector super-

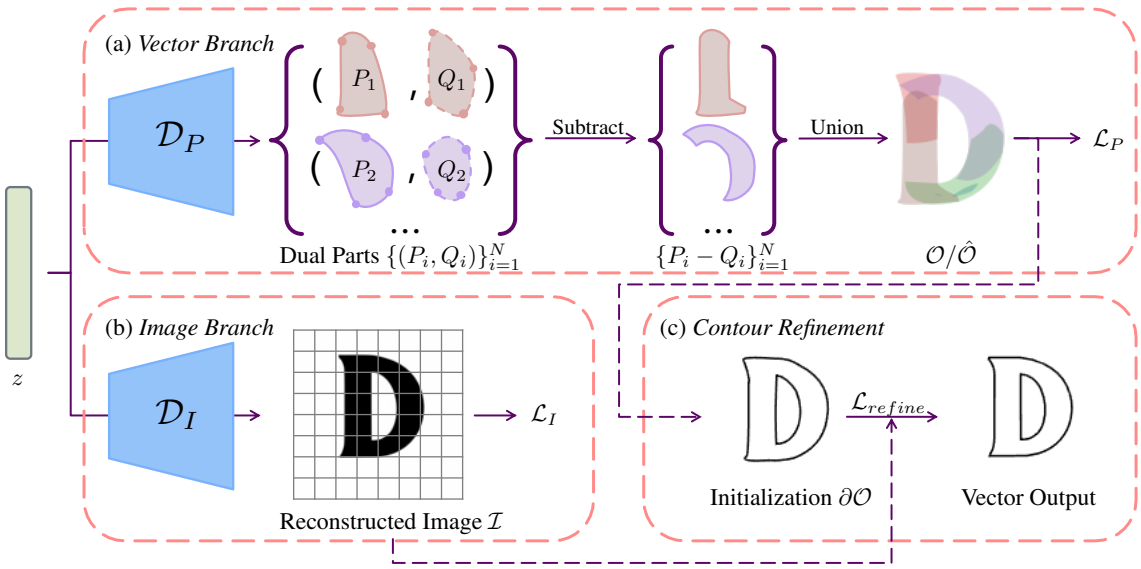

Figure 2. DualVector contains the following components: (a) a vector branch that maps the latent code z to several closed Bézier paths which are further gathered to form a global shape of the glyph; (b) an image branch that generates pixelated images with faithful details; (c) a contour refinement step that obtains the contour via boolean operations on dual-parts and optimizes it with the image guidance at inference time. Both the vector and the image branch are trained using the glyph images. Dual parts are distinguished by their colors.

vision. It rendered layered Bézier paths predicted from image features using DiffVG [22] and trains the system in an auto-encoding manner. DeepVecFont [39] adopted a dual-modality learning framework that utilizes both sequence and image features for high-quality vector font synthesis. But the results it produces may have the wrong topology, including self-intersection or invalid paths. In contrast to these methods, our approach does not require vector supervision and is very efficient in vector representation, with high quality glyph generation.

3. DualVector

In this section, we first illustrate the proposed dual-part representation for fonts (Sec. 3.1) and then introduce the components of DualVector for training this representation from glyph images (Secs. 3.2 to 3.4) and how the model can be used for font reconstruction and generation (Sec. 3.5). In DualVector, we adopt a joint representation of vector and image for fonts, as shown in Fig. 2. The latent space is shared by both modalities, and we associate each glyph with a latent code $z \in \mathbf{R}^d$ produced by the task-specific encoding process. The latent code z is fed into the two decoding branches to obtain a dual-part representation and an image representation respectively. The output of the vector branch is further processed by a contour refinement step to generate the final contour representation under the image guidance.

3.1. Dual-Part Representation

In our dual-part representation, we consider the closed parametric Bézier paths as our basic primitives for several reasons: (1) Bézier paths are powerful enough to approximate various shapes; (2) The contour of a closed shape produced by boolean operations on Bézier paths is still a set of Bézier paths, i.e., the closure property; (3) Bézier paths are widely used in modern font design, making the learned representation highly applicable. For convenience, we use a triplet of control points $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ to denote a quadratic Bézier curve $B(\cdot; \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ determined by the control points:

$$B(t; \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (1-t)^2\boldsymbol{a} + (1-t)t\boldsymbol{b} + t^2\boldsymbol{c}t \in [0,1] \quad (1)$$

Since a glyph often consists of a number of strokes with similar appearances, we propose to represent each glyph g with N closed Bézier paths $\{P_i\}_{i=1}^N$. Each path P_i is defined by M end-to-end connected quadratic Bézier curves $\{B_{i,j}(\boldsymbol{x}_{i,2j-1}, \boldsymbol{x}_{i,2j}, \boldsymbol{x}_{i,2j+1})\}_{j=1}^M$ $(\boldsymbol{x}_{i,2M+1} = \boldsymbol{x}_{i,1})$. We introduce the occupancy field of a closed path P_i as

$$\mathcal{O}_{P_i}(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in P_i \\ 0, & \boldsymbol{x} \notin P_i \end{cases} \quad (2)$$

The entire glyph shape g could simply be denoted as the union of all the paths $\cup_i P_i$. We normalize the coordinate range of the glyph to $[-1,1]^2$. Thus given any point $\boldsymbol{x}$ on the canvas $[-1,1]^2$, we can determine whether it is inside the represented glyph shape by a maximum operation, leading to the occupancy field of g:

$$\mathcal{O}(\boldsymbol{x}) = \max_i \mathcal{O}_{P_i}(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in \cup_i P_i \\ 0, & \boldsymbol{x} \notin \cup_i P_i \end{cases} \quad (3)$$

However, we find that the above representation can model fonts with convex curves on the contour but struggles to reconstruct shapes with holes. Therefore, in practice, we pair each path P_i with a negative path Q_i, and term this pair (P_i, Q_i) as a "dual part". In this way, g is represented as

$\cup_i(P_i - Q_i)$. The occupancy field $\mathcal{O}$ is then derived as:

$$\mathcal{O}(\boldsymbol{x}) = \max_i[\min(\mathcal{O}_{P_i}(\boldsymbol{x}), 1-\mathcal{O}_{Q_i}(\boldsymbol{x}))] = \begin{cases} 1, & \boldsymbol{x} \in g \\ 0, & \boldsymbol{x} \notin g \end{cases} \tag{4}$$

Even though $\mathcal{O}$ is the analytical occupancy field of g, it is not differentiable w.r.t. the parameters of the paths. Therefore, in order to apply gradient descent to learn such a representation, we calculate the approximate occupancy field $\hat{\mathcal{O}}$ from the signed distance field (SDF) of g. Since the distance $d(p; B)$ from any point p to a Bézier curve B can be derived analytically, the SDF of a path can be calculated differentiably w.r.t. its control points as follows:

$$s_{P_i}(\boldsymbol{x}) = [2\mathcal{O}_{P_i}(\boldsymbol{x}) - 1] \min_j d(\boldsymbol{x}; B_{i,j}) \tag{5}$$

Following previous vector graphics rasterization techniques [22, 31, 34], we approximate $\mathcal{O}$ by analytical pixel prefiltering with a parabolic kernel α:

$$\begin{aligned} \hat{\mathcal{O}}_{P_i}(\boldsymbol{x}) &= \alpha(s_{P_i}(\boldsymbol{x})) \\ \hat{\mathcal{O}} &= \max_i[\min(\hat{\mathcal{O}}_{P_i}, 1 - \hat{\mathcal{O}}_{Q_i})] \end{aligned} \tag{6}$$

3.2. Vector Branch

The vector branch takes a latent code z and outputs a set of dual-parts that represent a vector glyph. The path decoder $\mathcal{D}_P$ directly predicts the control points of the positive paths $\{P_i\}_{i=1}^N$ and the negative ones $\{Q_i\}_{i=1}^N$ from z:

$$\{\boldsymbol{x}_{i,j}\} = \mathcal{D}_P(z), 1 \le i \le 2N, 1 \le j \le 2M \tag{7}$$

where $\{\boldsymbol{x}_{i,j}\}_{i=1}^N$ are control points for $\{P_i\}_{i=1}^N$ and $\{\boldsymbol{x}_{i,j}\}_{i=N+1}^{2N}$ are control points for $\{Q_i\}_{i=1}^N$.

3.3. Image Branch

To combine with the advantages of image generation approaches, we train an auxiliary image branch that maps z to a pixelated image $\mathcal{I}$ of the glyph it represents. Here we adopt a CNN-based decoder $\mathcal{D}_I$:

$$\mathcal{I} = \mathcal{D}_I(z) \tag{8}$$

where $\mathcal{I}$ is a gray-scale image in shape $H \times W$. It provides detailed shape guidance to refine the vector shape contour.

3.4. Contour Refinement

Although the dual-part representation produced by the vector branch can already be rendered, there still exists a gap between it and modern contour representations for fonts. Therefore, we first convert it to the contour of the glyph $\partial\mathcal{O}$ with an off-the-shelf tool Paper.js [20] which can perform arbitrary boolean operations on SVG paths. Here the contour $\partial\mathcal{O}$ is a set of K closed Bézier paths.

$$\partial\mathcal{O} = \{C_1, C_2, ..., C_K\} \tag{9}$$

where K denotes the number of independent paths. Each individual part or hole increases K by 1. C_i is a closed Bézier path composed of l_i quadratic Bézier segments $\{\hat{B}_{i,j}(\hat{\boldsymbol{x}}_{i,2j-1}, \hat{\boldsymbol{x}}_{i,2j}, \hat{\boldsymbol{x}}_{i,2j+1})\}_{j=1}^{l_i}$.

After this conversion, we exploit the image generation results to further refine $\partial\mathcal{O}$ with differentiable rendering techniques [22]. $\mathcal{I}$ could naturally serve as a pixelated version of the occupancy field $\mathcal{O}$. During the refinement, we devise several strategies to improve the quality and efficiency of our representation:

Subdivision To improve the representational ability, we split long Bézier curves (with a length greater than 10% of the canvas width) at their midpoints.

Simplification To improve the efficiency of the contour representation, we replace the Bézier curves with a line segment connecting its two endpoints if they are close enough. For a Bézier curve $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, we simply define how faithfully the segment $\boldsymbol{ac}$ could represent the curve by the angle between $\vec{\boldsymbol{ba}}$ and $\vec{\boldsymbol{bc}}$. Also, we examine if there exist two adjacent Bézier segments that could be combined as a single one. Since all quadratic Bézier curves are parts of a parabola curve, we derive the parameters of the corresponding parabola curves from the control points and combine the two curves if their parameters are close enough. We also aggregate the endpoints of lines and curves that are too short.

Pruning Prior to refinement, some Bézier paths on the contour may enclose extremely small regions which will confuse the refinement process and produce redundant curves that eventually converge to a single point. We trim those closed paths C_i with areas under a certain threshold.

More implementation details of these strategies can be found in the supplemental material.

3.5. Implementation and Training

In this subsection, we will dive into how we utilize the proposed representation in both the font reconstruction and generation tasks.

3.5.1 Optimization Objectives

Optimization objectives can be divided into those at training time and those at inference time. The training-time objectives are task-related.

Font reconstruction. For the font reconstruction task, the input to our system is a gray-scale image of a glyph and the output is the corresponding vector representation. The pixelated image $\mathcal{I}_g$ of the glyph g is first passed through an image encoder $\mathcal{E}$ to obtain the latent code z:

$$z = \mathcal{E}(\mathcal{I}_g) \tag{10}$$

The latent code is then decoded by our vector branch and image branch into dual-parts $\{(P_i, Q_i)\}_{i=1}^N$ and a pixelated

output $\mathcal{I}$. We optimize the trainable modules $\mathcal{E}$, $\mathcal{D}_I$ and $\mathcal{D}_P$ simultaneously with the following loss functions:

$$\mathcal{L} = \lambda_P \mathcal{L}_P + \lambda_I \mathcal{L}_I \tag{11}$$

where we adopt an extra perceptual loss [43]:

$$\begin{aligned} \mathcal{L}_P &= \mathbb{E}_{\boldsymbol{x}\sim[-1,1]^2}[||\hat{\mathcal{O}}(\boldsymbol{x}) - \mathcal{I}_g(\boldsymbol{x})||_1] \\ \mathcal{L}_I &= ||\mathcal{I} - \mathcal{I}_g||_2 + \text{LPIPS}(\mathcal{I}, \mathcal{I}_g) \end{aligned} \tag{12}$$

Font generation. The input to the font generation task is multiple style reference images $\{R_i\}_{i=1}^{N_R}$ along with the corresponding character label T. The number of references N_R may vary during training to allow the model to accept a variable number of references during inference. The style references are first mapped to their features $\{f_i\}_{i=1}^{N_R}$ with an image encoder $\mathcal{E}_f$. We then apply stacked self-attention layers (SA) to aggregate these features to a style latent code z_{style} with a variational loss to enable font sampling. Character labels are mapped to vector representations by a learnable label embedding layer and then fused with z_{style} to yield the latent code z_T of the target glyph. Then z_T is fed to the decoders to generate the glyph, the same as the reconstruction task. The whole process can be described as follows:

$$\begin{aligned} f_i &= \mathcal{E}_f(R_i) \\ \mu, \sigma &= \text{SA}(f_1, f_2, ..., f_{N_R}) \\ z_{style} &= \mu + \epsilon\sigma, \epsilon \sim \mathcal{N}(0, 1) \\ z_T &= \text{FFN}([z_{style}, T]) \end{aligned} \tag{13}$$

To ease training, we use the pre-trained $\mathcal{E}$ from the reconstruction task to encode ground truth output g_T and train the encoder with the following guidance:

$$\mathcal{L}_{latent} = ||z_T - \mathcal{E}(g_T)||_2 \tag{14}$$

To enable font sampling, we adopt a Kullback-Leibler divergence loss [16]:

$$\mathcal{L}_{kl} = \text{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, 1)) \tag{15}$$

Next, the encoder is fine-tuned along with the pre-trained $\mathcal{D}_I$ and $\mathcal{D}_P$ from the reconstruction task using the loss in 11 with $\mathcal{L}_{kl}$.

Refinement Both the reconstruction and generation tasks share the same contour refinement process. Given $\partial\mathcal{O}$ and $\mathcal{I}$, the refinement process produces the optimized glyph contour in SVG format by inference-time optimization. In the optimization, we utilize DiffVG [22], denoted as DR, to render the contour in a differentiable manner. The control points of $\mathcal{O}$ are refined to minimize the following loss function:

$$\mathcal{L}_{refine} = \mathcal{L}_{ras} + \lambda_{reg}\mathcal{L}_{reg} \tag{16}$$

where $\mathcal{L}_{render}$ is the photometric loss between the rasterized image and the image branch output.

$$\mathcal{L}_{ras} = ||\text{DR}(\partial\mathcal{O}) - \mathcal{I}||_1 \tag{17}$$

The regularization term limits the overall length of $\mathcal{O}$.

$$\mathcal{L}_{reg} = \sum_{i=1}^{K} \text{len}(C_i) \tag{18}$$

3.5.2 Unsigned Distance Field Warm-up

Although the vector branch is end-to-end trainable, the gradients only appear near the boundaries. Therefore, to overcome the vanishing gradient problem, we add an extra loss based on the unsigned distance field (UDF) of the vector parts to warm up the system training.

In general, for two closed curves c_1, c_2, we cannot obtain the exact SDF of the shape after arbitrary boolean operations on the curves by point-wise operations on their respective SDFs. For the union of two SDFs, as an approximation, we have

$$s_{c_1\cup c_2}(\boldsymbol{x}) \begin{cases} = \min(s_{c_1}(\boldsymbol{x}), s_{c_2}(\boldsymbol{x})), & \boldsymbol{x} \notin c_1 \cup c_2 \\ \leq \min(s_{c_1}(\boldsymbol{x}), s_{c_2}(\boldsymbol{x})), & \boldsymbol{x} \in c_1 \cup c_2 \end{cases} \tag{19}$$

The $\min(\cdot)$ operation gives a correct SDF outside the shape but an inaccurate SDF inside the shape. Similarly, the $\max(\cdot)$ gives a correct SDF inside and an inaccurate SDF outside for the intersection of two SDFs. Therefore, it is hard to calculate the accurate SDF $s_{\mathcal{O}}$ for the glyph. Fortunately, we only aim to provide a coarse initialization for the dual parts, preventing the optimization process from falling into local minima early in the training. We find that using an approximate unsigned distance field works well to achieve this goal. Here we define the UDF u as

$$u(\boldsymbol{x}) = \max(0, s(\boldsymbol{x})) \tag{20}$$

where $\min(\cdot)$ gives a correct global UDF for unions of shapes. The approximate UDF $\hat{u}_g$ is computed as

$$\hat{u}_g = \min_i[\max(u_{P_i}, u_{Q_i})] \tag{21}$$

where

$$\begin{aligned} u_{P_i}(\boldsymbol{x}) &= \max(s_{P_i}(\boldsymbol{x}), 0) \\ u_{Q_i}(\boldsymbol{x}) &= \max(-s_{Q_i}(\boldsymbol{x}), 0) \end{aligned} \tag{22}$$

We apply the following losses to warm up the training process for the font reconstruction task:

$$\begin{aligned} \mathcal{L} &= \lambda_P \mathcal{L}_P + \lambda_I \mathcal{L}_I + \lambda_u \mathcal{L}_u \\ \mathcal{L}_u &= \mathbb{E}_{\boldsymbol{x}\sim[-1,1]^2}[||\hat{u}_g(\boldsymbol{x}) - u_g(\boldsymbol{x})||_1] \end{aligned} \tag{23}$$

where the ground truth UDF u_g is derived from the pixel image g.

4. Experiments

4.1. Dataset

We use the SVG-Fonts dataset from SVG-VAE [27] for all the experiments. Following DeepVecFont [39], we sample a subset of the dataset with 8035 fonts for training and 1425 fonts for evaluation. The resolution of input glyph images is set to 128×128. Please refer to the supplementary material for implementation details.

4.2. Font Reconstruction

In this experiment, we investigate how faithfully different methods can reconstruct the input glyph images with

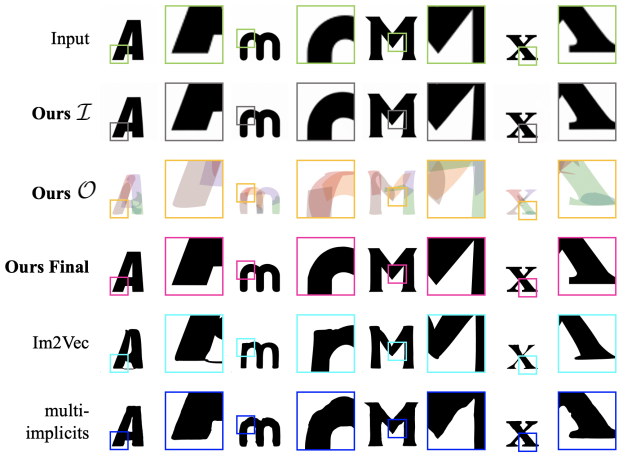

Figure 3. Comparison of font reconstruction quality. We show the input glyph image row by row, our reconstructed image $\mathcal{I}$ from the image branch, dual-parts $\mathcal{O}$ from the vector branch, and our final result after contour refinement, as well as vector outputs from alternative methods.

vector outputs, which can also be regarded as font vectorization. Im2Vec [33] and our method directly produce vector representations from glyph images, while for Multi-Implicits [34], we adopt the 2D marching cube algorithm and extract the contours. As can be seen from a qualitative comparison in Fig. 3, our representation could faithfully reconstruct global shapes as well as local details from the pixelated input. Im2Vec [33] could only preserve coarse shapes and suffers from wrong topologies as shown in the "A" example as it is easy to fall into local optimum without the global gradient. Multi-Implicits [34] results in finer details than Im2Vec [33] but still exhibit unsmooth edges and rounded corners. To quantitatively evaluate how accurate the vector outputs are, we calculate three metrics, namely SSIM, L1, and s-IoU, between the rendered images and the ground truth images under different resolutions. Tab. 1 shows that our method surpasses the two alternatives by a clear margin on all metrics. Using parametric curves as primitives, our method, along with Im2Vec [33], maintains a stable L1 error across all resolutions for the smooth edges, while for Multi-Implicits [34] the L1 error gets larger as the resolution increases, resulting from the high-frequency noises at the extracted boundaries.

4.3. Font Generation

To evaluate the generation ability of our method, we adopt the few-shot font generation setting, where a subset of glyphs with the same style is given and models are trained to produce a complete set of glyphs that are consistent with the input glyphs in style. This task is also dubbed "font complement" or "font style transfer" in other works. We compare our method with several popular methods in font generation: DeepVecFont [39], Multi-Implicits [34], Im2Vec [33] and Attr2Font [38]. For a fair comparison, we use the same four characters, 'A', 'B', 'a', and 'b' as style references for each font and evaluate image-level metrics on the generated characters. Since Im2Vec [33] is an image vectorization method, we apply it to the images generated by Attr2Font [38]. Multi-Implicits [34] follows an auto-decoder architecture, so we freeze the decoder and find the optimal latent vector for a font by minimizing the losses between the given style references and the predicted ones using gradient descent.

Table 1. Quantitative comparison with Im2Vec [33] and Multi-Implicits [34] on three image-level metrics at different resolutions for the font reconstruction task. The gray scale is normalized to [0,1]. s-IoU, from [34], measures the overlap.

Method	Resolution	SSIM↑	L1↓	s-IoU↑
Ours	128^2	**0.9436**	**0.0137**	**0.8879**
	256^2	**0.9483**	**0.0136**	**0.9068**
	512^2	**0.9579**	**0.0136**	**0.9135**
Multi-Implicits	128^2	0.9231	0.0183	0.8709
	256^2	0.9262	0.0208	0.8721
	512^2	0.9419	0.0213	0.8739
Im2Vec	128^2	0.7800	0.0504	0.6832
	256^2	0.8566	0.0504	0.6847
	512^2	0.8957	0.0504	0.6851

As illustrated in Fig. 4, DeepVecFont [39] may generate characters with wrong topologies due to the incorrect initial contour prediction, such as 'A', 'L', and 'Z'. The fixed topology in the optimization process makes it difficult to produce the necessary details, which are particularly symbolic in serif fonts. In addition, unconstrained contour generation may also produce self-intersections, like the 'g' and 'r', which are not easy to fix in subsequent processing. Im2Vec [33] tends to produce rounded corners and is only able to produce the coarse style, lacking details. Multi-Implicits [34] generates vector results from implicit signed distance fields (SDF) through a 2D Marching cube algorithm, leading to free-form contours. Therefore the edges are not smooth enough and have unpleasing noises, such as the 'g' and 'x' in the figure. Attr2Font [38] can generate generally satisfactory font images, but it cannot maintain high-quality rendering when scaling.

Compared with the above methods, our method is capable of not only grasping the overall style of the input references but also generating new glyphs with clear boundaries and sharp edges, achieving optimal visual quality. The efficient dual-part representation along with the differentiable occupancy supervision establishes a link between pixelated images and vector graphics, allowing the unsupervised synthesis of vector fonts with aligned shapes and correct topologies. The refinement step allows the composition of the contour to be freely changed and thus ensures

Figure 4. Font generation results of Attr2Font, Im2Vec, Multi-Implicits, DeepVecFont, and DualVector. The input style references are marked with red boxes. We trace the zero-level sets of SDFs in a piece-wise linear way to get SVGs from Multi-Implicits [34]. Typical failure cases are marked in blue dashed boxes. Some details are zoomed-in in the leftmost column.

Table 2. Quantitative comparison of image-level metrics of Attr2Font [38], Im2Vec [33], Multi-Implicits [34], DeepVecFont [39] and DualVector in the font complement experiment.

Method	Resolution	SSIM↑	L1↓	s-IoU↑
Ours	128^2	**0.8288**	**0.0461**	**0.7395**
	256^2	**0.8727**	**0.0460**	**0.7453**
	512^2	**0.9075**	**0.0460**	**0.7469**
DeepVecFont	128^2	0.8165	0.0506	0.7125
	256^2	0.8640	0.0506	0.7171
	512^2	0.9011	0.0506	0.7182
Multi-Implicits	128^2	0.8025	0.0578	0.6857
	256^2	0.8533	0.0603	0.6814
	512^2	0.8914	0.0608	0.6811
Im2Vec	128^2	0.7576	0.0733	0.5731
	256^2	0.8270	0.0734	0.5751
	512^2	0.8739	0.0734	0.5756
Attr2Font[a]	128^2	0.7841	0.0512	0.6393

[a] [38] synthesizes pixelated images only.

that our synthesized glyphs have high-fidelity details under the guidance of high-resolution glyph images. Quantitative results on three image-level metrics at different resolutions are shown in Tab. 2. For methods producing vector shapes, we also count the average number of commands used for each synthesized vector character, with smaller numbers indicating more compact outputs. The results are shown in Tab. 3. Multi-Implicits [34] uses a large number of short line segments to represent the boundary due to the marching cube algorithm. Im2Vec [33] represents a shape with 4 parts each enclosed by 20 cubic Bézier curves, lacking flexibility and details. Our method performs boolean operations on the quadratic Bézier curves from dual parts, achieving comparable compactness with human-designed fonts and outputs from methods with vector supervision [39].

Table 3. The average number of commands used per glyph for different methods in the font generation task. M, L, Q, and C denote move, straight line, quadratic Bézier curve, and cubic Bézier curve respectively.

Command	M	L	Q	C	Total↓
Ours	1.42	8.21	11.06	0	20.69
DeepVecFont [39]	1.37	8.23	0	9.68	**19.28**
Multi-Implicits [34]	6.16	1446.53	0	0	1452.69
Im2Vec [33]	4	0	0	80	84
Human-Designed	1.38	9.17	0	8.51	19.06

4.4. Ablation Studies

To demonstrate the effectiveness of several key design choices in our method, we conduct ablation studies on the font reconstruction task. We investigate the following degraded variants: (a) replace the union of dual parts with the union of all the paths; (b) train without the UDF warm-up process; and (c) remove the refining strategy during refinement. We show the comparison results of a representative case in Fig. 5. Without the dual-part representation, variant

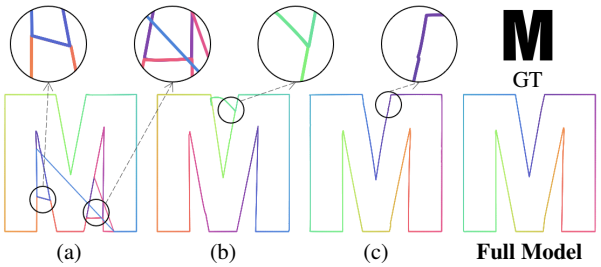

Figure 5. The contours of an "M" reconstructed by our method, compared with three degraded variants.

Table 4. Quantitative results of reconstruction accuracy for different model variants in ablation studies. All the output vector glyphs are rendered at 512×512 to compute the L1 error.

	L1↓	#Lines↓	# Curves↓
(a) w/o dual parts	0.0143	12.23	14.06
(b) w/o warm-up	0.0146	11.19	13.12
(c) w/o refining strategy	0.0148	-	22.07
Full Model	**0.0136**	**8.97**	**11.81**

(a) cannot reconstruct concave regions very well, as they can be hard to represent by the union of simply closed paths. Without UDF initialization, variant (b) produces contours with useless curves that are hard to be eliminated during optimization. Without the refining strategy, variant (c) may generate inaccurate edges with zigzags. Quantitative results in Tab. 4 also verify the quality improvements brought by the three design choices.

We also experimentally investigate the effect of different (N, M)s on the representation capacity. We choose several settings of (N, M) and train only the vector branch (i.e. no contour refinement) with the same number of epochs to reconstruct the input. Tab. 5 shows that $M = 4$ is optimal and the reconstruction accuracy of the vector branch increases as N increases. The accuracy does not gain too much changing N from 6 to 8. So considering the time cost for training and inference, we select $(N, M) = (6, 4)$ in the above experiments.

Table 5. Different reconstruction settings and metrics. The image-based metrics use a resolution of 512×512.

N	M	SSIM↑	L1↓	s-IOU↑
2	4	0.9262	0.0282	0.8338
4	4	0.9292	0.0259	0.8464
6	4	0.9303	0.0250	0.8512
8	4	0.9306	0.0247	0.8525
6	3	0.9293	0.0256	0.8480
6	5	0.9300	0.0252	0.8498

4.5. Font Sampling and Interpolation

In the font generation task, the style latent code z_{style} is trained with a variational loss $\mathcal{L}_{kl}$, enabling generating fonts of new styles by sampling z_{style} from $\mathcal{N}(0, 1)$. We show in Fig. 1 (left) and Fig. 6 multiple styles of fonts sampled. Please refer to our supplementary material for more examples. Also, benefiting from our dual-part representation, we can perform smooth interpolation between arbitrary font styles as demonstrated in Fig. 1 (right).

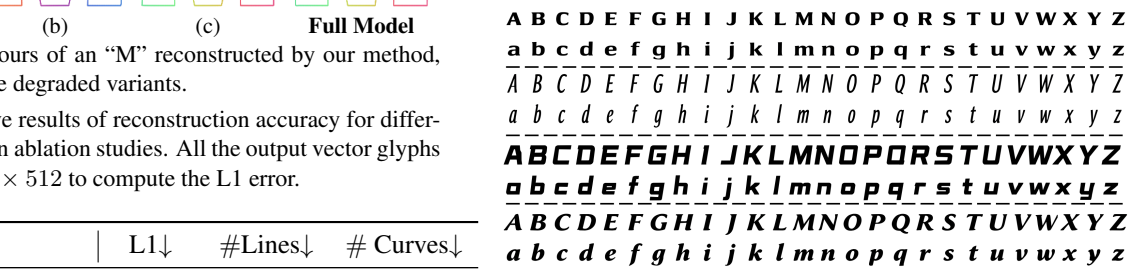

Figure 6. New fonts with style codes sampled in $\mathcal{N}(0, 1)$.

5. Limitations

The contour refinement step requires gradient descent during inference and therefore imposes a large time overhead, similar to DeepVecFont [39]. The process could potentially be accomplished by forwarding inferences of some generation models, such as the diffusion models [13]. Another limitation is that DualVector only focuses on the synthesis of glyphs on a fixed-size canvas, without considering the kerning between them, making the spacing between characters less natural. This may be solved by post-processing the SVG with some automatic kerning methods or by learning through a data-driven approach.

6. Conclusion

We present DualVector as a new representation and generation method for vector fonts. DualVector models a glyph as dual parts and introduces a contour refinement step to add pleasing details to the contours of the dual parts. Our method is able to generate vector fonts without vector supervision, which can be directly converted to common digital font formats for content presentation and distribution. Comparisons with different methods of font reconstruction and generation tasks demonstrate that our approach produces fonts with the best quality among all the alternatives. In the future, more plausible initialization and supervision could be exploited to generalize this representation to more complex fonts (e.g. Chinese fonts). We hope this research contributes to the typeface design industry and provide inspiration for broader works on the understanding and generation of 2D man-made content such as cartoon characters and icons.

Acknowledgement This work was supported by the Natural Science Foundation of China (Project Number 61832016), and Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology.

References

[1] Adobe Inc. Adobe illustrator. 2

[2] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 11, page 13, 2018. 1, 2

[3] Vineet Batra, Mark J. Kilgard, Harish Kumar, and Tristan Lorach. Accelerating vector graphics rendering using the graphics hardware pipeline. *ACM Trans. Graph.*, 34(4), jul 2015. 2

[4] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1, 2

[5] Cedar Lake Ventures, Inc. Vector magic. 2

[6] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 42–51. IEEE, 2020. 2

[7] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5939–5948. Computer Vision Foundation / IEEE, 2019. 1

[8] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey E. Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 31–41. IEEE, 2020. 2

[9] A. E. Fabris and A. R. Forrest. Antialiasing of curves by discrete pre-filtering. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 317–326, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 2

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014. 2

[11] Chris Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*, pages 9–18. 2007. 2

[12] Hideaki Hayashi, Kohtaro Abe, and Seiichi Uchida. Glyphgan: Style-consistent font generation based on generative adversarial networks. *Knowl. Based Syst.*, 186, 2019. 1, 2

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 8

[14] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017. 2

[15] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Dcfont: an end-to-end deep chinese font generation system. In Diego Gutierrez and Hui Huang, editors, *SIGGRAPH Asia 2017 Technical Briefs, Bangkok, Thailand, November 27 - 30, 2017*, pages 22:1–22:4. ACM, 2017. 2

[16] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, pages 1–14, 2014. 2, 5

[17] Yoshiyuki Kokojima, Kaoru Sugita, Takahiro Saito, and Takashi Takemoto. Resolution independent rendering of deformable vector objects using graphics hardware. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, page 118–es, New York, NY, USA, 2006. Association for Computing Machinery. 2

[18] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4):99:1 – 99:8, 2011. 2

[19] Gregory Lecot and Bruno Levy. Ardeco: Automatic region detection and conversion. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, page 349–360, Goslar, DEU, 2006. Eurographics Association. 2

[20] Jürg Lehni and Jonathan Puckey. Paper.js: The swiss army knife of vector graphics scripting. https://github.com/paperjs/paper.js, 2021. 4

[21] Rui Li, Qiming Hou, and Kun Zhou. Efficient gpu path rendering using scanline rasterization. *ACM Trans. Graph.*, 35(6), nov 2016. 2

[22] Tzu-Mao Li, Michal Lukác, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph.*, 39(6):193:1–193:15, 2020. 2, 3, 4, 5

[23] Xiang Li, Lei Wu, Xu Chen, Lei Meng, and Xiangxu Meng. Dse-net: Artistic font image synthesis via disentangled style encoding. In *ICME*, pages 1–6. IEEE, 2022. 2

[24] Xianming Lin, Jie Li, Hualin Zeng, and Rongrong Ji. Font generation based on least squares conditional generative adversarial nets. *Multim. Tools Appl.*, 78(1):783–797, 2019. 1

[25] Ying-Tian Liu, Yuan-Chen Guo, Yi-Xiao Li, Chen Wang, and Song-Hai Zhang. Learning implicit glyph shape representation. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–12, 2022. 2

[26] Charles Loop and Jim Blinn. Resolution independent curve rendering using programmable graphics hardware. In *ACM SIGGRAPH 2005 Papers*, pages 1000–1009. 2005. 2

[27] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *2019 IEEE/CVF International Conference*

on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019, pages 7929–7938. IEEE, 2019. 1, 2, 5

[28] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, aug 1987. 2

[29] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2022. 2

[30] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV (1)*, volume 12346 of *Lecture Notes in Computer Science*, pages 405–421. Springer, 2020. 1

[31] Diego Nehab and Hugues Hoppe. Random-access rendering of general vector graphics. *ACM Trans. Graph.*, 27(5):135, 2008. 2, 4

[32] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, pages 165–174. Computer Vision Foundation / IEEE, 2019. 1, 2

[33] Pradyumna Reddy, Michaël Gharbi, Michal Lukác, and Niloy J. Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 7342–7351. Computer Vision Foundation / IEEE, 2021. 1, 2, 6, 7

[34] Pradyumna Reddy, Zhifei Zhang, Zhaowen Wang, Matthew Fisher, Hailin Jin, and Niloy J. Mitra. A multi-implicit neural representation for fonts. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 12637–12647, 2021. 1, 2, 4, 6, 7

[35] Peter Selinger. Potrace: a polygon-based tracing algorithm. 2003. 2

[36] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11–es, jul 2007. 2

[37] Yuchen Tian. zi2zi: Master chinese calligraphy with conditional adversarial networks. https://github.com/kaonashi-tyc/zi2zi, 2017. 2

[38] Yizhi Wang, Yue Gao, and Zhouhui Lian. Attribute2font: Creating fonts you want from attributes. *ACM Trans. Graph.*, 39(4), July 2020. 2, 6, 7

[39] Yizhi Wang and Zhouhui Lian. Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Trans. Graph.*, 40(6):265:1–265:15, 2021. 1, 3, 5, 6, 7, 8

[40] Yankun Xi, Guoli Yan, Jing Hua, and Zichun Zhong. Jointfontgan: Joint geometry-content GAN for font generation via few-shot learning. In *ACM Multimedia*, pages 4309–4317. ACM, 2020. 2

[41] Shuai Yang, Jiaying Liu, Wenjing Wang, and Zongming Guo. Tet-gan: Text effects transfer via stylization and destylization. In *AAAI Conference on Artificial Intelligence*, 2019. 1, 2

[42] Gao Yue, Guo Yuan, Lian Zhouhui, Tang Yingmin, and Xiao Jianguo. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Trans. Graph.*, 38(6), 2019. 1, 2

[43] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 586–595. Computer Vision Foundation / IEEE Computer Society, 2018. 5

[44] Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):618–629, 2009. 2

[45] Shuang Zhao, Frédo Durand, and Changxi Zheng. Inverse diffusion curves using shape optimization. *IEEE Trans. Vis. Comput. Graph.*, 24(7):2153–2166, 2018. 2

[46] Haikuan Zhu, Juan Cao, Yanyang Xiao, Zhonggui Chen, Zichun Zhong, and Yongjie Jessica Zhang. Tcb-spline-based image vectorization. *ACM Trans. Graph.*, 41(3):34:1–34:17, 2022. 2