Joint Implicit Neural Representation for High-fidelity and Compact Vector Fonts

Chia-Hao Chen
Tsinghua University
jh-chen20@mails.tsinghua.edu.cn

Ying-Tian Liu
Tsinghua University
liuyingt23@mails.tsinghua.edu.cn

Zhifei Zhang
Adobe Research
zzhang@adobe.com

Yuan-Chen Guo
Tsinghua University
guoyc19@mails.tsinghua.edu.cn

Song-Hai Zhang
Tsinghua University
shz@tsinghua.edu.cn

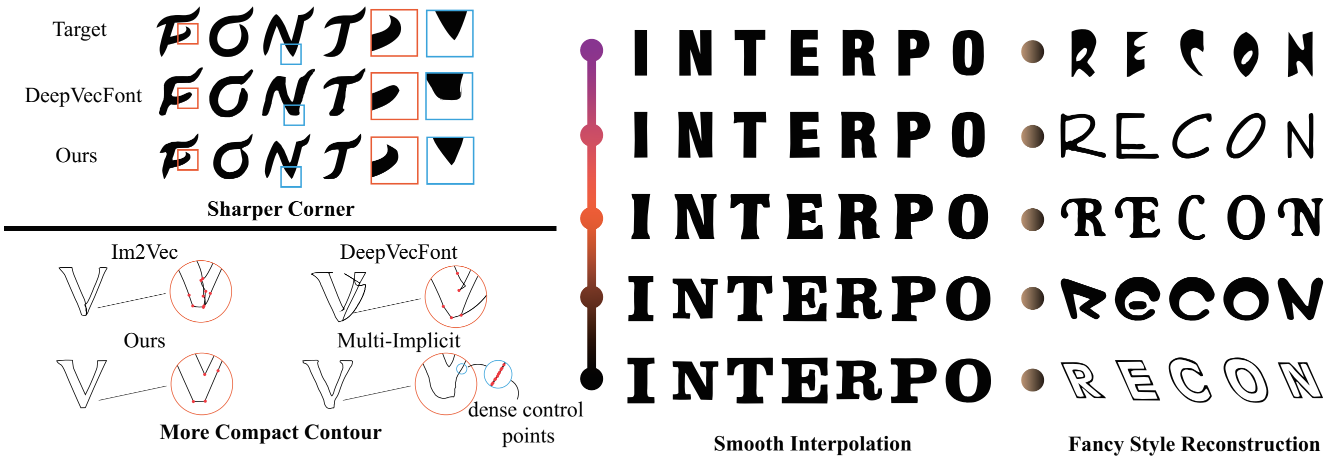

Figure 1: Overview of our vector font generation results. Our approach reconstructs font shapes with sharper corners and more compact contours while enabling smooth interpolation between different font styles.

Abstract

Existing vector font generation approaches either struggle to preserve high-frequency corner details of the glyph or produce vector shapes that have redundant segments, which hinders their applications in practical scenarios. In this paper, we propose to learn vector fonts from pixelated font images utilizing a joint neural representation that consists of a signed distance field (SDF) and a probabilistic corner field (CF) to capture shape corner details. To achieve smooth shape interpolation on the learned shape manifold, we establish connections between the two fields for better alignment. We further design a vectorization process to extract high-quality and compact vector fonts from our joint neural representation. Experiments demonstrate that our method can generate more visually appealing vector fonts with a higher level of compactness compared to existing alternatives.

1. Introduction

The automatic generation of stylized fonts has important applications in art and design. The vector font is preferred to the pixelated font image by virtue of its scalability in rendering and compactness in storage and therefore has become mainstream.

Existing vector font generation methods either directly work on the vector format [3, 25, 34], or perform vectorization on the generated pixelated font image [26]. The former mainly represents a glyph shape as a sequence of Scalable Vector Graphic (SVG) drawing commands and learns to generate such sequences by ground truth vector supervision or utilizing a differentiable vector shape renderer [15]. Despite the ability of this representation for capturing fine shape details, these methods suffer from the ambiguity brought by the nature of the many-to-one mapping from the drawing commands to the rendered shape, in which case different drawing commands could produce the same-looking shape. This ambiguity often leads to redundant segments in the generated sequence, degrading shape quality. The latter stream benefits from coordinate-based neural implicit fields that emerged in recent years [4, 21], and tends to produce higher-quality overall shape. However, the inherent smoothness of the network output with respect to the input coordinates hinders these methods from accurately modeling high-frequency geometric details like the corners.

Based on the above observations, in this paper, we aim to present an implicit neural representation of fonts that can both model geometric details and be used to synthesize high-quality vector fonts. Our intuition is to enhance existing implicit neural representations with explicit modeling of corner positions and design a procedure to convert this representation to vector primitives. More specifically, we model the overall glyph shape using a 2D signed distance field (SDF), and model a probabilistic corner field (CF) to indicate the probability of each position being a corner point. We learn a latent space of the proposed representation on a large number of fonts using their pixelated images and ground truth corner positions as supervision. To achieve smoothness interpolation in the latent space, we build correspondences of the two fields by a novel signed distance flow loss. Converting this joint implicit neural representation to vector fonts is not trivial, as we need to integrate the corners into the base shapes without bringing artifacts. To achieve this, we take inspiration from the dual contouring algorithm [13] to locate the coordinates of corners from CF, and obtain a compact vector shape via incremental curve fitting. We compare our method with existing vector font/shape synthesis methods and demonstrate that our proposed representation can achieve higher quality in the font reconstruction and interpolation task, while being able to generate compact vector output.

In summary, our contributions are as follows:

- A joint neural implicit representation for fonts that combines a signed distance field (SDF) and a probabilistic corner field (CF) to capture corner details of glyph shapes better.
- A customized vectorization process to convert the joint representation to compact vector fonts.
- State-of-the-art quality on font reconstruction and interpolation task both qualitatively and quantitatively.

2. Related work

2.1. Pixel Font Synthesis

Many works on font synthesis adopted Generative Adversarial Network (GAN) [8] to generate diverse stylish character [31, 35]. Also, there are some other encoder-decoder-based approaches that are able to output high-quality calligraphy [12, 37, 11]. And a few works conducted few-shot learning to perform style transfer [2, 7, 24]. Despite the strength of GAN, it usually suffers from unstable training [27] and mode collapse [30], so finding an efficient and stable solution for generative task counts for diversely stylish glyph generation. Other than GAN, the representation based on SDF [16, 17, 36] leverages the power of implicit field also reaches competitive results in multiple tasks.

2.2. Vector Font Synthesis

In many cases, fonts are stored in vector representations, which are scalable and require much less storage space than pixmaps, so a lot of work is also aimed at generating vector representations. By vectorizing pixmaps, Im2Vec [25] allows converting pixel fonts to vector fonts, while LIVE [20] converts more complex images to SVG images with better preservation of image detail and topology structures. SVG-VAE [18] sequential autoencoder architecture that extracts SVG commands to reconstruct fonts. DeepSVG [3] utilizes the power of the transformer to obtain these SVG commands. DeepVecFont [34] introduces multi-modal learning based on the structure of SVG-VAE, and refine the SVG on the pixel image to obtain more accurate reconstruction result. Reddy et al. [26] presents a multi-curve implicit representation to reproduce sharp corners. DualVector [17] employs unsupervised learning to acquire font shape through multiple implicit dual components, subsequently refining these components on a learned pixelated image to achieve a concise vector contour. VecFontSDF [36] adopts parabolic shape primitives, with a more precise SDF supervision signal, and achieves high reconstruction and synthesis quality. In addition, DiffVG [15] makes vector curves differentiable, establishing relationships between vector graphics and bitmaps, which allows them to be optimized by the network and gradually become a theoretical foundation of many vector shape generation works.

2.3. Neural Implicit Representation

Implicit neural representation (INR) yield appealing result in view synthesis such as Neural Radiance Field (NeRF) [22] and shape representation like Distance Field [23, 5] and Occupancy Field [21]. NeuS [32] combines the previous two, estimates the object's geometry through volume rendering, and can restore the object's surface texture. Compared with convolution, implicit representation has better position representation and interpolation properties [4]. By encoding the coordinates instead of pixel value, it would mitigate the texture sticking [14]. For a Signed Distance Field (SDF) representation, it's easy to extract its contour or surface via marching cube [19], marching tetrahedra[28] or dual contouring [13].

In our work, we represent the font by two implicit fields, SDF and CF, with HyperNetworks. The SDF part will use RELU as the activation function, and CF will use sine as

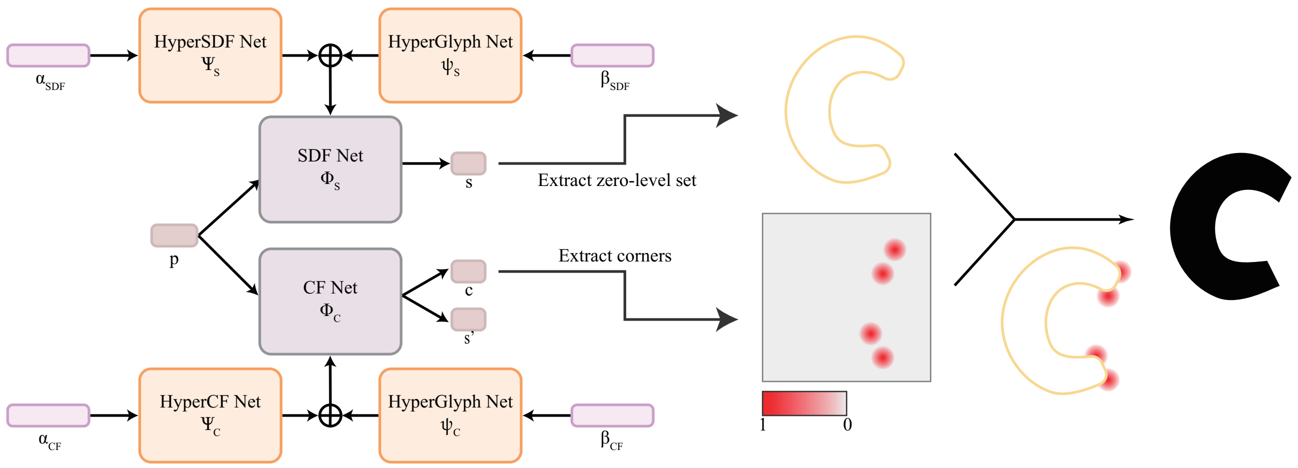

Figure 2: Network architecture and vectorization process of our proposed method. For any vector shape, the network inputs the latent code of its font and glyph and outputs the SDF and CF values of the corresponding pixel coordinates. Then we extract the contour of the shape by the zero-level set of SDF, the position of the corners by the peak of CF, and finally, combine these results to generate vector fonts by our rendering method.

activation and be initialized as proposed in [29].

3. Method

In this section, we will introduce a joint implicit representation consisting of two neural fields: a signed distance field (SDF) and a corner field (CF) for fonts. In sec.Sec. 3.1 and Sec. 3.2, we train two neural networks, namely the SDF network and the CF network, to represent our fonts. The SDF network takes the coordinate of pixels p, an integer $I_{\text{glyph}} \in [0, N_{\text{char}})$ and an integer $I_{\text{style}} \in [0, N_{\text{font}})$ as inputs, and outputs the SDF value of the corresponding font, where $N_{\text{char}} = 52$ is the number of characters from A-z, and N_{font} is the number of font style groups in training data. On the other hand, the CF network takes p, I_{glyph}, and I_{style} as inputs. It outputs the CF value and an auxiliary SDF value dubbed as SDFlow. These two networks are trained independently with different training objectives, L_{SDF} and L_{CF}.

Then we align the glyph shapes implied by the two fields in Sec. 3.3 and finally we describe a vectorization process to extract high-quality vector fonts in Sec. 3.4. We disentangle the latent representation for each glyph to the font-style-related and the character-structure-related codes. So we associate each glyph with four learn-able latent codes, $\alpha_{\text{SDF}}, \beta_{\text{SDF}}, \alpha_{\text{CF}}, \beta_{\text{CF}}$, in which $\alpha_{\text{SDF}} \in \mathbb{R}^{D_1}, \alpha_{\text{CF}} \in \mathbb{R}^{D_3}$ is for font styles and $\beta_{\text{SDF}} \in \mathbb{R}^{D_2}, \beta_{\text{CF}} \in \mathbb{R}^{D_4}$ is for character structures. These codes can be retrieved by selecting the I_{style}-th and I_{glyph}-th vectors from the respective codebook utilized in the neural networks.

3.1. Signed Distance Field

In this part, we use an SDF to represent the corner-free shape of the glyph. To increase the model capacity, we adopt two HyperNetworks [10, 6], Ψ_S, ψ_S to consume the latent codes $\alpha_{\text{SDF}}, \beta_{\text{SDF}}$ and produce the parameters of the SDF network Φ_S. More specifically, the parameters of this SDF network θ are the element-wise sum of the outputs of the two HyperNetworks:

$$\theta = \Psi_S(\alpha_{\text{SDF}}) + \psi_S(\beta_{\text{SDF}}) \tag{1}$$

where $\Psi_S : \mathbb{R}^{D_1} \to \mathbb{R}^m, \psi_S : \mathbb{R}^{D_2} \to \mathbb{R}^m$ and m is the number of the parameters of the SDF network. In our implementation, Φ_S takes the query coordinate p as the input and predicts the signed distance value at p, following a ReLU-activated MLP architecture. We adopt L1 loss for SDF reconstruction:

$$\mathcal{L}_{\text{SDF}} = \|\bar{s} - \Phi_S(p;\theta)\|_1 \tag{2}$$

where $\bar{s}$ is the ground truth SDF value at coordinate p. The hyper-network-based approach usually learns a better latent space than directly predicting the signed distance conditioned on a latent code with an MLP with fixed parameters and is experimentally proved to have finer fitting results.

Since the ground truth normal value $\bar{n}$ could be easily obtained by parametric curves (i.e. Bézier curves or lines), we can apply the contour constraint loss to supervise the normal, i.e. the derivatives of the SDF values with respect to the input coordinates, for a better contour:

$$\mathcal{L}_{\text{Normal}} = 1 - \langle \nabla_p \Phi_S(p;\theta), \bar{n} \rangle \tag{3}$$

where ∇ is the spatial gradient of SDF, and $\langle \cdot, \cdot \rangle$ denote the cosine similarity of two vectors. Like many other works using implicit SDFs [9, 1], we also use Eikonal loss to constrain the correctness of the SDF:

$$\mathcal{L}_{\text{Eikonal}} = |\|\nabla_p \Phi_S(p;\theta)\|_2 - 1| \tag{4}$$

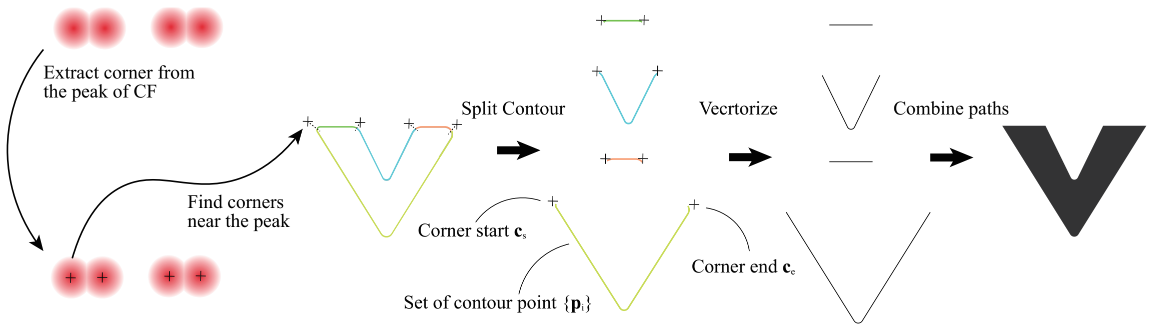
Extract corner from the peak of CF
Find corners near the peak
Split Contour
Vecrtorize
Combine paths
Corner start $\mathbf{c}_s$
Corner end $\mathbf{c}_e$
Set of contour point $\{\mathbf{p}_i\}$

Figure 3: Vectorization process. First, we find the peaks from the corner point field as possible corner positions, then we find the nearest points to each peak on the contour contour points (zero-level set) obtained by SDF, and use these points to split the contour. Each set (including the first two corner points and a section of contour points) is vectorized by curve fitting, and the beginning and end control points of the vector curve are ensured to be on the corner points. Finally, the vector font is obtained by combining these vector curves.

In summary, the SDF network is trained with the following objective:

$$L_{\text{SDF}} = \lambda_{\text{SDF}}\mathcal{L}_{\text{SDF}} + \lambda_{\text{Normal}}\mathcal{L}_{\text{Normal}} + \lambda_{\text{Eikonal}}\mathcal{L}_{\text{Eikonal}} \quad (5)$$

where λ_*'s are the balancing weights for different loss terms.

3.2. Corner Field

Because SDF struggles to model high-frequency details such as corners, which are decisive in terms of the visual quality and usability of fonts, we introduce another implicit field called Corner Field (CF). Generally, it is difficult to encode extremely sparse information in the network, such as a binary image with only a few pixels in white. So for a corner, it can't only have a value of 1 in its position and a value of 0 in other places. Therefore, we choose to set the corner field as a probability field, like the Gaussian Distribution in DARK [38]. It intuitively reflects the probability that a corner point exists in the neighborhood of a query point p. But different from DARK, which separates each joint part and represents it independently, we model all corners in a single field with the target CF defined as:

$$\bar{c}(p) = \max_i \exp\left(-\frac{\|p - c_i\|_2^2}{\xi^2}\right) \quad (6)$$

where ξ controls the range of the probability field, c_i is the position of the i-th corner of this glyph. It is necessary to adopt this global representation because the number of corners of each glyph is not fixed, some characters may have as many corners as much more than 20, and some glyphs may even have no corner, in this case, $c(p) = 0$.

Similar to SDF, we adopt another two HyperNetworks Ψ_C, ψ_C to form the parameters of the CF network Φ_C.

$$\omega = \Psi_C(\alpha_{\text{CF}}) + \psi_C(\beta_{\text{CF}}) \quad (7)$$

where $\Psi_C : \mathbb{R}^{D_3} \rightarrow \mathbb{R}^l, \psi_C : \mathbb{R}^{D_4} \rightarrow \mathbb{R}^l$ and l is the number of the parameters of the CF network. Φ_C is a sine-activated [29] MLP that consumes the query coordinate p and produces the corner field value c at p along with an auxiliary signed distance value s':

$$c, s' = \Phi_C(p; \omega) \quad (8)$$

We will discuss s' later in Sec. 3.3.

Since CF uses a probabilistic representation, we use the binary cross-entropy loss as our reconstruction loss:

$$\mathcal{L}_{\text{CF}} = \text{BCE}(\Phi_C(p; \omega).c, \bar{c}) \quad (9)$$

In addition, the output of our networks is a field, which means we need a mechanism for locating the corners. By Eq. (6) we know that the extrema of the field are corners. Visually, we could tell corners are located in the middle of the field, or in other words, the peak of the field. So we propose the peak loss for peak enhancement:

$$\mathcal{L}_{\text{Peak}} = \bar{c}(p)^2 \mathcal{L}_{\text{CF}} \quad (10)$$

In summary, the CF network is trained with the following objective:

$$L_{\text{CF}} = \lambda_{\text{CF}}\mathcal{L}_{\text{CF}} + \lambda_{\text{Peak}}\mathcal{L}_{\text{Peak}} + \lambda_{\text{SDFlow}}\mathcal{L}_{\text{SDFlow}} \quad (11)$$

where λ_*'s represent the balancing weights for different loss terms, and $\mathcal{L}_{\text{SDFlow}}$ serves as an additional supervision of SDF value at each p. This supervision is crucial for interpolation and will be further explored and discussed in Sec. 3.3.

3.3. Alignment of SDF and CF

Our final goal is not only to reconstruct the glyph but also to find a good latent space for interpolation. Naturally, SDF bears an interpolation property because the change of the SDF value at the queried coordinate describes exactly the movement of the glyph contour while the CF is not necessarily of this nature. Therefore, we add an auxiliary SDF supervision to the CF network to constrain the network parameters:

$$\mathcal{L}_{\text{SDFlow}} = \|\bar{s} - \Phi_C(p).s'\|_1 \tag{12}$$

where $\bar{s}$ is the ground truth SDF value at p. For the last linear layer of Φ_C, we have:

$$\begin{aligned} s' &= W_s h \\ c &= \sigma(W_c h) \end{aligned} \tag{13}$$

where $\sigma(\cdot)$ is the sigmoid function, W_c, W_s is the weights related to output c and s respectively of the last layer of Φ_C, and h is the hidden output of the penultimate layer network. We expect to relate changes in s' and c in this way, and in Fig. 4, we also demonstrate the validity of the alignment. During interpolation, the CF without $\mathcal{L}_{\text{SDFlow}}$ suffers from peaks vanishing in intermediate results, and those peaks would emerge in the final results, while CF with $\mathcal{L}_{\text{SDFlow}}$ would provide a rational peak transfer corresponding to the glyph shape.

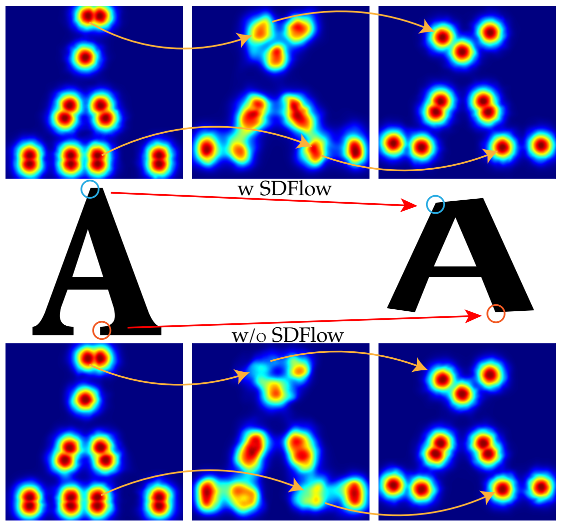

Figure 4: Interpolation w, w/o $\mathcal{L}_{\text{SDFlow}}$, corresponding corner in fonts where make differences on CF are marked with circles.

3.4. Vectorization Process

To generate vector graphics that are scalable, such as SVG or TTF format fonts, we do not stop at the joint representation but will continue to produce functional vector fonts from the two implicit fields.

Extract contour points. The first step of vectorization is to transfer the zero-level set obtained by SDF into an ordered point set on the glyph contour. We start by converting

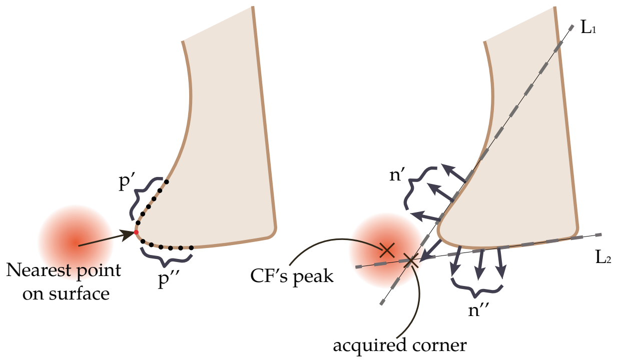

Figure 5: Locate the corner from CF by dual contouring.

the field into a binary pixel image I that:

$$I(p) = \begin{cases} 0, & \Phi_S(p/R) \geq 0 \\ 1, & \Phi_S(p/R) < 0 \end{cases} \tag{14}$$

where $p = (x, y) \in \{1, 2, ..., R\}^2$ is the pixel coordinate on the image, R is the resolution of the image to be rendered, then we perform the following edge detection:

$$M = (1 - I) \cdot (I \otimes \mathbf{1}) \tag{15}$$

where $\otimes$ is convolution operator and $\mathbf{1}$ is a 3×3 convolution kernel matrix filled with 1. $p(x, y)$ is on the contour if and only if $M(x, y) = 1$. We will then locate the corner points. And finally, we use incremental curve fitting to vectorize the glyph given the contour points and corner points.

Locate corners with Dual Contouring. After extracting the ordered points on the contour from SDF, we need to determine the position of corners from CF, which usually served as the start and end control points of a path. In CF, each peak corresponds to a surrounding corner. Since CF is a probabilistic representation, the expected corner point does not coincide exactly with the peak. So when we find the peak, we need to locate a corner in its neighborhood. Similar to dual contouring [13], we choose the tangent intersection of two points on the contour near the peak as the corner. When we find the point closest to the peak, we should find two tangent lines on both sides of it, as shown in Fig. 5. We identify these two lines, $\mathbf{L}_1$ and $\mathbf{L}_2$, by the following method:

$$\begin{cases} \mathbf{L}_1 = \sum_{i=0}^{n} C_\lambda(k)\mathbf{p}'_i + t(\sum_{i=0}^{n} C_\lambda(k)\mathbf{n}'_i)_\perp \\ \mathbf{L}_2 = \sum_{i=0}^{n} C_\lambda(k)\mathbf{p}''_i + t(\sum_{i=0}^{n} C_\lambda(k)\mathbf{n}''_i)_\perp \end{cases} \tag{16}$$

where $\mathbf{p}', \mathbf{p}''$ are pixel coordinates in both sides of the contour point nearest to the corner, ordered from near to far,

$\mathbf{n}', \mathbf{n}''$ are corresponding contour normal vectors on $\mathbf{p}', \mathbf{p}''$, and $(\cdot)_\perp$ denotes the vector perpendicular to the vector inside. The $C_\lambda(k)$ above is defined by:

$$C_\lambda(k) = \frac{e^{-\lambda}\lambda^k}{k!} / \sum_{i=0}^{n} \frac{e^{-\lambda}\lambda^i}{i!} \tag{17}$$

which is actually a normalized coefficient of Possion distribution. Since the points near the corner points will have smooth outputs in the network, we should intuitively care about the coordinates and tangents of the points a little farther away from the corner points, while we do not care about the points further away either, and the Poisson distribution fits this data distribution modality. So we adopt the Poisson distribution to weigh the coordinates and tangent vectors of the contour points near the corner point to form the tangent lines. The final corner is the intersection of $\mathbf{L}_1$ and $\mathbf{L}_2$.

Incremental Curve Fitting. After obtaining the corners and contour, we split them into multiple sets of $\{\mathbf{c}_s, \mathbf{c}_e, \{\mathbf{p}_i\}\}$, as shown in Fig. 3. For any $\{\mathbf{c}_s, \mathbf{c}_e, \{\mathbf{p}_i\}\}$, we need to find several curves or lines as a path that takes $\mathbf{c}_s$ as the start control point, and $\mathbf{c}_e$ as the end control point, and minimize the fitting error of $\{\mathbf{p}_i\}$, $i = 0, ..., n$.

From the first point of $\{\mathbf{p}_i\}$, we maintain a set $\mathbb{S} = \varnothing$ initially, constantly adding the point into $\mathbb{S}$. Initially, we take the $\mathbf{p}_0$ as the start control point of the splines we want to fit. Progressively, each time we add the temporary $\mathbf{p}_i$ in, we consider it to be the end control point of the straight line $\mathbf{L}$ we want to fit, and we calculate the maximum error of the distance from the point in $\mathbb{S}$ to the line. If it exceeds a threshold $l_{\min}$, there are two different cases according to the length of the line:

1. If $\|\mathbf{L}\|_2 \geq l_{\min}$, record $\mathbf{L}$, empty $\mathbb{S}$, and fit the next spline.
2. If $\|\mathbf{L}\|_2 < l_{\min}$, we consider the points in $\mathbb{S}$ are on a quadratic Beziér curve $\mathbf{Q}$, and start fitting this curve.

For the quadratic Bézier curve, since we have determined the start and end control points, the only thing left to do is to determine the second control point. In addition, we know that there are no corners between $\mathbf{c}_s$ and $\mathbf{c}_e$, so the splines we fit must be smoothly connected. We utilize the good property of the Bézier curve that its tangent on the start or end point is in the same direction as the connection between it and the adjacent control point. Thereby, we just need to find the second control point along the tangent direction of the control point at the end of the previous curve, and here we can use linear searching (for example, place the second control point at small intervals from the starting point, then calculate the error, and finally take the position with the smallest error as the position of the second control point), or more efficiently, binary search. The detail is described in the supplementary material. Finally, we add $\mathbf{c}_s$ and $\mathbf{c}_e$ to the spline curve as the start and end points.

4. Experiments

In this section, we conduct experiments on the font shape reconstruction and interpolation task in Sec. 4.1 and Sec. 4.2. We first compare with several related methods, including Im2Vec [25], Multi-Implicit [26], DeepVecFont [34] and Attr2Font [33]. Im2Vec is a vector shape generation method that learns from pixelated images. DeepVecFont and Multi-Implicit are state-of-the-art vector font synthesis methods. DeepVecFont directly learns drawing command sequences while Multi-Implicit models font shape with implicit neural fields and utilizes an isosurface extraction method to get vector outputs. Attr2Font is a font generation method that works on the image domain. We perform the quantitative comparison on L1, SSIM, and s-mIoU, and also show qualitative comparison results. We demonstrate the compactness of our generated vector font in Sec. 4.3 and perform ablation studies to evaluate the effectiveness of some key design choices in Sec. 4.4. We used the test set data from DeepVecFont [34] to train all the methods for fair comparisons. It contains 1425 fonts, each having 52 glyphs from A to z.

4.1. Reconstruction

We tested the reconstruction accuracy of each method at different resolutions with L1 error, Structural Similarity Index (SSIM), and soft mean Intersection of Union (s-mIoU). The ground truth data was rasterized from the vector paths, and we translated each letter so that they were centered on the image. We did not perform the centering for DeepVecFont and instead used its own data preprocessing procedure.

As shown in Tab. 1, our model outperforms other methods, which demonstrates that our proposed representation has enough capacity to model these shapes. Without CF, details of font shapes will be missing and corners are oversmoothed. This is more clearly illustrated in Fig. 1 (Sharper Corner) rather than the table, where CF promotes extra details for complex shapes. Fig. 9 demonstrates some reconstruction results in the dataset.

4.2. Interpolation

Interpolation between existing fonts is an important means of generating new fonts. Although there are various possibilities for the interpolation between any two font styles, our ultimate goal is to make the interpolation result match the font distribution as closely as possible as is mentioned in Multi-Implicit [26]. We sample 1000 fonts from the dataset, group them into 500 pairs, and interpolate between fonts in each pair with an interpolation factor of

Methods	L1↓	SSIM↑	s-mIoU↑
Ours	**.0140/.0136/.135**	**.9672/.9682/.9657**	**.9264/.9272/.9299**
Ours w/o CF	.0160/.0153/.0154	.9610/.9631/.9581	.9194/.9219/.9238
Im2Vec	.0332/.0341/.0339	.9055/.9172/.9003	.8657/.8475/.8254
DeepVecFont	.0418/.0419/.0419	.8509/.8515/.8451	.7310/.7303/.7317
Multi-Implicit	.0408/.0401/.0402	.8853/.8880/.8875	.8161/.8192/.8187
Attr2Font	.1057/.1066/.1064	.7143/.7186/.7183	.6075/.6042/.6046

Table 1: Reconstruction results in different resolutions ($128^2/256^2/512^2$).

Methods	L1↓	SSIM↑	s-mIoU↑
Ours	.0485/.0489/.0488	**.8804**/.8800/.8775	**.8106/.8095/.8103**
Im2Vec	.0649/.0658/.0658	.8203/.8186/.8158	.7250/.7222/.7230
DeepVecFont	**.0454/.0456/.0456**	.8466/.8464/.8408	.7217/.7205/.7217
Multi-Implicit	.0710/.0711/.0711	.8230/.8236/.8234	.7476/.7479/.7478
Attr2Font	.0514/.0528/.0527	.8796/**.8829/.8826**	.8023/.7950/.7958

Table 2: Interpolation results ifferent resolutions ($128^2/256^2/512^2$).

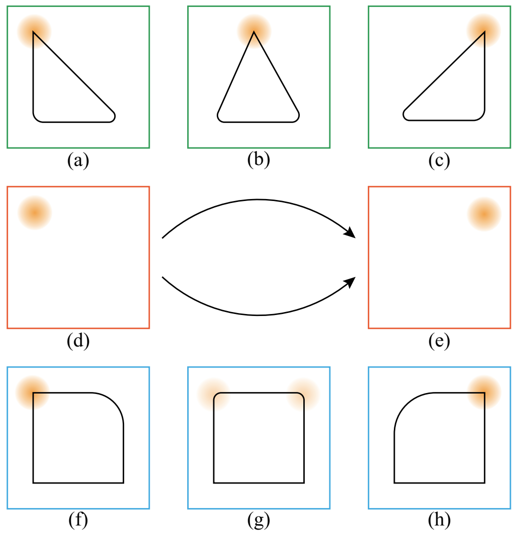

Figure 6: Multiple correspondences of corner field transfer.

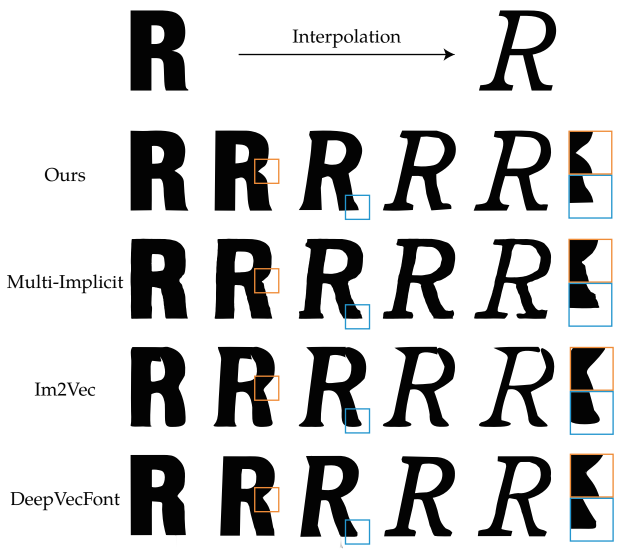

Figure 7: Corner preservation in interpolation: For the two fonts to be interpolated, the intermediate interpolation font should have corners consistently in the parts they have corners.

0.5. For each interpolation result, we find the corresponding fonts with the smallest L1 and largest SSIM and s-mIoU respectively from the original dataset as the pseudo ground truth to compute the final metrics. As indicated in Tab. 2, our method achieves overall the best performance among all competitors. In Fig. 7 we show that our method preserves the sharp corners well during interpolation.

It is worth mentioning that since we adopt an auto-decoder architecture, the latent codes for each font are retrieved by optimization on frozen network parameters. However, fitting the latent code for our corner field representation brings ambiguity, as shown in Fig. 6: There are two ways to transfer the corner field from (d) to (e), from the (a) to (c) or from the (f) to (h). The CF in (a) and (f) share different latent codes while their CF looks the same. Therefore, like Im2Vec, we generate different font shapes by interpolation.

Despite these limitations, the SDF network demonstrates capability in few-shot style transfer and shape completion tasks. By freezing the network parameters, we optimize a style code using the SDF value derived from the target shape, in conjunction with the glyph code supplied by the pre-trained model as the HyperNetworks' input. Detailed information and corresponding results will be presented in the supplementary material.

Methods	M	L	Q	C	CMDs	CPs
Ours	1.40	12.01	18.83	0	32.25	51.08
DeepVecFont	1.43	9.10	0	12.21	22.75	**47.17**
Im2Vec	c	0	0	20c	**21c**	61c
Multi-Implicit	5.68	1681	0	0	1686	1686
Human-Designed	1.43	9.04	0	9.56	20.03	39.17

Table 3: Number of Commands (CMDs) and Control Points (CPs). c is the color parameter in Im2Vec, and we adopt $c = 1$ with only black color in experiment.

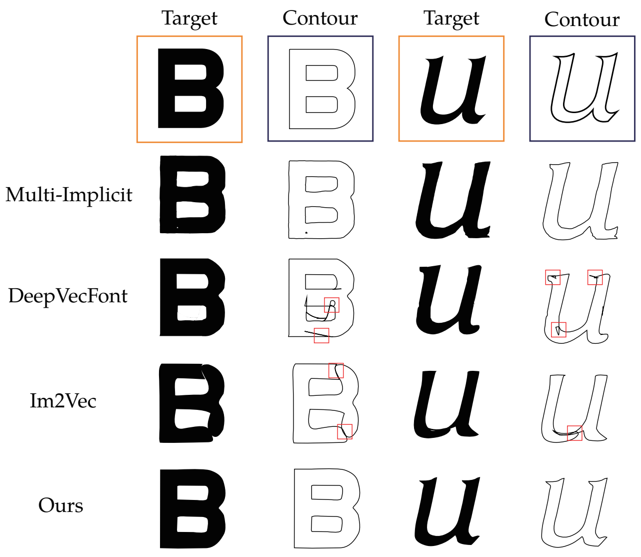

Figure 8: The comparison of compactness of different methods, our method uses a small number of curves and there is no redundancy and crossover between curves.

4.3. Compactness of Vector

Although several methods involved in the comparison can generate vector fonts, there are significant differences in the way they are generated. What these methods have in common is that they all generate a series of path sequences (includes path consisting of M, L, Q, and C commands in SVG, where M and L have 1 control point, Q has 2 and C has 3 control points), but the number of the commands of these sequences is different, as shown in the Tab. 3. Different methods generate vector fonts that have different numbers of path commands (CMDs) and control points (CPs). It can be seen that DeepVecFont and Im2Vec have advantages in these two aspects, but they sometimes introduce wrong structures, as shown in Fig. 8 (red box), while the method of vectorizing zero level set with an implicit representation does not have these problems and has a tight surface. On top of that, our method has a comparable number of commands and path points as DeepVecFont and Im2Vec, resulting in more compactness.

4.4. Ablation study

In this section, we first conduct ablation studies on the loss function proposed in the Corner Field, then we train several models of SDF Net with or without HyperNetworks to verify its importance.

Level	0.3	0.5	0.7	0.9
w/o $\mathcal{L}_{\text{Peak}}$	1.183	.6832	.6833	.6832
w $\mathcal{L}_{\text{Peak}}$	1.109	.5806	.5877	.5877

Table 4: Error of the number of corners between network outputs w, w/o $\mathcal{L}_{\text{Peak}}$ and GT.

Peak Constraint. When two corners are too close, the peaks of the corner field obtained by the network may stick together, resulting in the existence of only one peak. Tab. 4 shows the L1 error between the corner number of ground truth and network output with or without the peak loss. The level represents that when the value of a pixel (probability between 0∼1) is greater than its 8 neighbor pixels, as well as greater than a threshold equal to that level, it'll be treated as a corner, otherwise not. We use a default level of 0.5 in our reconstruction and interpolation experiments. It can be seen that when training without peak loss, the disparity between the number of output corners and the number of ground truth corners will significantly increase.

SDF Constraint. The signed distance flow can be used to constrain the interpolation results of the corner field so as to obtain a reasonable latent space of the corner field corresponding to the SDF results because we lack the ground truth for intermediate results. Because of this, we need metrics other than L1 or SSIM to measure the quality of the interpolation results with or without the signed distance flow. In most situations, the shape transfer would lead to a corner transfer from (a)-(c) rather than (f)-(h) in Fig. 6, such as the example in Fig. 4. Therefore, we apply the chamfer distance between interpolated corners C_I and the corners to interpolate C_0 and C_1 to measure their divergence, which we expect to be a relatively large value:

$$\text{Div}(\uparrow) = \frac{1}{\|c_I\|} \min\Big(\sum_{i=0}^{c_I} \min_{j=0}^{c_0}(\|C_I[i] - C_0[j]\|), \sum_{i=0}^{c_I} \min_{j=0}^{c_1}(\|C_I[i] - C_1[j]\|)\Big) \tag{18}$$

where c_0, c_1, c_I is the number of corners in C_0, C_1, C_I, and $C_x[y]$ denote the y-th corner in C_x. Results shows a total Div $=$ 971.7 without $\mathcal{L}_{\text{SDFlow}}$ and Div $=$ 988.6 with

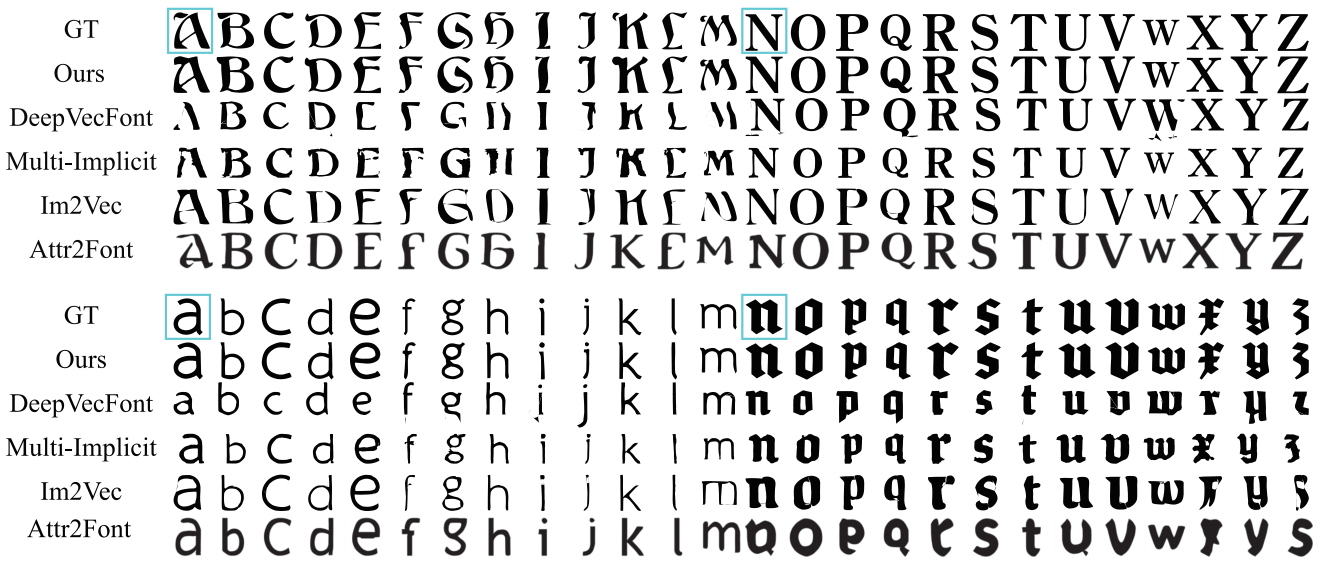

Figure 9: Different methods comparison on several reconstructed fonts. The cyan box marks the starting glyph of each font style.

$\mathcal{L}_{\text{SDFlow}}$, which demonstrates the effectiveness of the SDF constraint in interpolation.

HN L/F	IN L/F	L1(↓)	SSIM(↑)	s-mIoU(↑)
4/384	5/256	.0154	.9581	.9238
4/384	3/256	.0210	.9437	.9030
3/384	5/256	**.0151**	**.9594**	**.9264**
4/256	5/256	.0177	.9432	.8972
-	5/256	.0455	.8620	.7892
-	5/384	.0354	.8924	.8287
-	8/256	.0408	.8843	.8206
-	8/384	.0277	.9175	.8634

Table 5: Results on different network configuration w and w/o HyperNetworks(HN).

HyperNetworks. To establish the indispensability of HyperNetworks (HN), we conducted training on multiple SDF networks, both with and without the implementation of HN. Among them, four models incorporate HN while the other four models exclude it. The latter groups of models without HN utilized the concatenation of α_S, β_S, and coordinate p as input, bypassing the adoption of HyperNetworks for parameter prediction and instead directly training the SDF Net.

Tab. 5 shows these results on the reconstruction of HyperNetworks, under the 512^2 resolution. HN refers to the HyperNetworks Ψ_S and ϕ_S, while IN denotes the SDF Net Φ_S. L/F denotes the number of layers/feature dimensions of the MLP in HN and IN. The first group denotes the default configuration we adopt in the Sec. 4.1 and Sec. 4.2. The results demonstrate that HyperNetworks significantly enhances the accuracy of font representation. In terms of training efficiency, the 6th group (the 2nd group w/o HN) has a similar training duration as the first one, while its performance is much inferior to the one that has HyperNetworks. Furthermore, the last group requires more than half of the additional training overhead compared to the first group, yet there is still a discernible performance gap between them. However, the comparison between the 1st and the 3rd groups reveals that augmenting the network parameters does not consistently contribute to reconstruction. Thus, there's ample room for improvement in the architecture design of HyperNetworks. Detailed illustrations of more network architecture designs will be provided in the supplementary materials.

5. Conclusion

We present a novel font representation for improved shape details. It consists of a signed distance field (SDF) and a probabilistic corner field (CF) that models the number and positions of the corners. Based on the proposed representation, we propose a vectorization procedure to combine the SDF and CF to generate high-quality vector fonts with compact contours. We hope our work can inspire new directions for integrating high-frequency details in areas like 3D shape modeling and generation. The biggest limitation of this work is that it cannot randomly sample a group of corresponding SDF and CF from the networks, which could be solved by a more powerful encoder mixing the modal of two fields and embedding them into the same latent space.

References

[1] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574, 2020. 3

[2] Samaneh Azadi, Matthew Fisher, Vladimir G 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*, pages 7564–7573, 2018. 2

[3] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, 2020. 1, 2

[4] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 2

[5] Julian Chibane, Gerard Pons-Moll, et al. Neural unsigned distance fields for implicit function learning. *Advances in Neural Information Processing Systems*, 33:21638–21652, 2020. 2

[6] Yu Deng, Jiaolong Yang, and Xin Tong. Deformed implicit field: Modeling 3d shapes with learned dense correspondence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10286–10296, 2021. 3

[7] Yue Gao, Yuan Guo, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019. 2

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 2

[9] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020. 3

[10] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 3

[11] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Dcfont: an end-to-end deep chinese font generation system. In *SIGGRAPH Asia 2017 Technical Briefs*, pages 1–4. 2017. 2

[12] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Scfont: Structure-guided chinese font generation via deep stacked networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4015–4022, 2019. 2

[13] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002. 2, 5

[14] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34:852–863, 2021. 2

[15] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. 1, 2

[16] 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*, 2022. 2

[17] Ying-Tian Liu, Zhifei Zhang, Yuan-Chen Guo, Matthew Fisher, Zhaowen Wang, and Song-Hai Zhang. Dualvector: Unsupervised vector font synthesis with dual-part representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14193–14202, 2023. 2

[18] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7930–7939, 2019. 2

[19] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 2

[20] 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/CVF Conference on Computer Vision and Pattern Recognition*, pages 16314–16323, 2022. 2

[21] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 2

[22] 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. *Communications of the ACM*, 65(1):99–106, 2021. 2

[23] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 2

[24] Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Few-shot font generation with localized style representations and factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2393–2402, 2021. 2

[25] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7342–7351, 2021. 1, 2, 6

[26] Pradyumna Reddy, Zhifei Zhang, Zhaowen Wang, Matthew Fisher, Hailin Jin, and Niloy Mitra. A multi-implicit neural representation for fonts. *Advances in Neural Information Processing Systems*, 34:12637–12647, 2021. 1, 2, 6

[27] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques

for training gans. *Advances in neural information processing systems*, 29, 2016. 2

[28] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021. 2

[29] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 3, 4

[30] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. *Advances in neural information processing systems*, 30, 2017. 2

[31] Yuchen Tian. zi2zi: Master chinese calligraphy with conditional adversarial networks. *https://github. com/kaonashi-tyc/zi2zi*, 3, 2017. 2

[32] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021. 2

[33] Yizhi Wang, Yue Gao, and Zhouhui Lian. Attribute2font: Creating fonts you want from attributes. *ACM Transactions on Graphics (TOG)*, 39(4):69–1, 2020. 6

[34] Yizhi Wang and Zhouhui Lian. Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021. 1, 2, 6

[35] Shan-Jean Wu, Chih-Yuan Yang, and Jane Yung-jen Hsu. Calligan: Style and structure-aware chinese calligraphy character generator. *arXiv preprint arXiv:2005.12500*, 2020. 2

[36] Zeqing Xia, Bojun Xiong, and Zhouhui Lian. Vecfontsdf: Learning to reconstruct and synthesize high-quality vector fonts via signed distance functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1848–1857, 2023. 2

[37] Yangchen Xie, Xinyuan Chen, Li Sun, and Yue Lu. Dgfont: Deformable generative networks for unsupervised font generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5130–5140, 2021. 2

[38] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. Distribution-aware coordinate representation for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7093–7102, 2020. 4