VecFontSDF: Learning to Reconstruct and Synthesize High-quality Vector Fonts via Signed Distance Functions

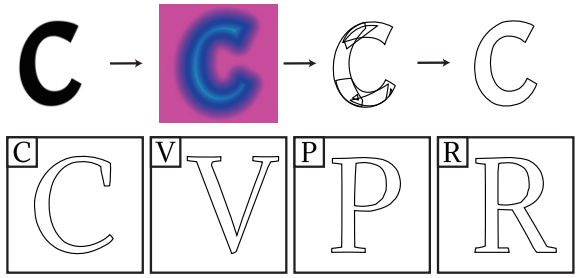
Zeqing Xia*, Bojun Xiong*, Zhouhui Lian†
Wangxuan Institute of Computer Technology, Peking University, China

Abstract

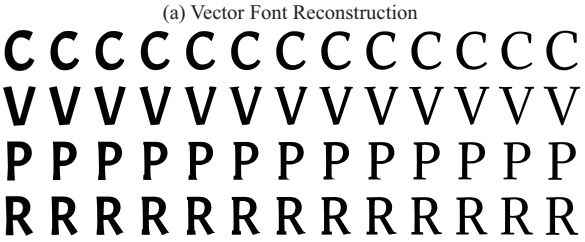
Font design is of vital importance in the digital content design and modern printing industry. Developing algorithms capable of automatically synthesizing vector fonts can significantly facilitate the font design process. However, existing methods mainly concentrate on raster image generation, and only a few approaches can directly synthesize vector fonts. This paper proposes an end-to-end trainable method, VecFontSDF, to reconstruct and synthesize high-quality vector fonts using signed distance functions (SDFs). Specifically, based on the proposed SDF-based implicit shape representation, VecFontSDF learns to model each glyph as shape primitives enclosed by several parabolic curves, which can be precisely converted to quadratic Bézier curves that are widely used in vector font products. In this manner, most image generation methods can be easily extended to synthesize vector fonts. Qualitative and quantitative experiments conducted on a publicly-available dataset demonstrate that our method obtains high-quality results on several tasks, including vector font reconstruction, interpolation, and few-shot vector font synthesis, markedly outperforming the state of the art.

1. Introduction

Traditional vector font designing process relies heavily on the expertise and effort from professional designers, setting a high barrier for common users. With the rapid development of deep generative models in the last few years, a large amount of effective and powerful methods [1, 7, 32] have been proposed to synthesize visually-pleasing glyph images. In the meantime, how to automatically reconstruct and generate high-quality vector fonts is still considered as a challenging task in the communities of Computer Vision and Computer Graphics. Recently, several methods



(a) Vector Font Reconstruction



(b) Vector Font Interpolation

Figure 1. Examples of results obtained by our method in the tasks of vector font reconstruction (a) and vector font interpolation (b).

based on sequential generative models [3, 11, 20, 30, 33] have been reported that treat a vector glyph as a draw-commands sequence and use Recurrent Neural Networks (RNNs) or Transformer [31] to encode and decode the sequence. However, this explicit representation of vector graphics is extremely difficult for the learning and comprehension of deep neural networks, mainly due to the long-range dependence and the ambiguity in how to draw the outlines of glyphs. More recently, DeepVecFont [33] was proposed to use dual-modality learning to alleviate the problem, showing state-of-the-art performance on this task. Its key idea is to use a CNN encoder and an RNN encoder to extract features from both image and sequence modalities. Despite using richer information of dual-modality data, it still needs repetitively random samplings of synthesis results to find the optimal one and then uses Diffvg [18] to refine the vector glyphs under the guidance of generated images in the inference stage.

Another possible solution to model the vector graphic is to use implicit functions which have been widely used to represent 3D shapes in recent years. For instance, DeepSDF [25] adopts a neural network to predict the values of signed distance functions for surfaces, but it fails to con-

*Denotes equal contribution.

†Corresponding author. E-mail: lianzhouhui@pku.edu.cn

This work was supported by National Language Committee of China (Grant No.: ZDI135-130), Center For Chinese Font Design and Research, and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).

vert those SDF values to the explicit representation. BSP-Net [4] uses hyperplanes to split the space in 2D shapes and 3D meshes, but it generates unsmooth and disordered results when handling glyphs consisting of numerous curves.

Inspired by the above-mentioned existing methods, we propose an end-to-end trainable model, VecFontSDF, to reconstruct and synthesize high-quality vector fonts using signed distance functions (SDFs). The main idea is to treat a glyph as several primitives enclosed by parabolic curves which can be translated to quadratic Bézier curves that are widely used in common vector formats like SVG and TTF. Specifically, we use the feature extracted from the input glyph image by convolutional neural networks and decode it to the parameters of parabolic curves. Then, we calculate the value of signed distance function for the nearest curve of every sampling point and leverage these true SDF values as well as the target glyph image to train the model.

The work most related to ours is [19], which also aims to provide an implicit shape representation for glyphs, but possesses a serious limitation mentioned below. For convenience, we name the representation proposed in [19] IGSR, standing for "Implicit Glyph Shape Representation". The quadratic curves used in IGSR are not strictly parabolic curves, which cannot be translated to quadratic Bézier curves. As a consequence, their model is capable of synthesizing high-resolution glyph images but not vector glyphs. Furthermore, it only uses raster images for supervision which inevitably leads to inaccurate reconstruction results. On the contrary, our proposed VecFontSDF learns to reconstruct and synthesize high-quality vector fonts that consist of quadratic Bézier curves by training on the corresponding vector data in an end-to-end manner. Major contributions of our paper are threefold:

- We design a new implicit shape representation to precisely reconstruct high-quality vector glyphs, which can be directly converted into commonly-used vector font formats (e.g., SVG and TTF).
- We use the true SDF values as a strong supervision instead of only raster images to produce much more precise reconstruction and synthesis results compared to previous SDF-based methods.
- The proposed VecFontSDF can be flexibly integrated with other generative methods such as latent space interpolation and style transfer. Extensive experiments have been conducted on these tasks to verify the superiority of our method over other existing approaches, indicating its effectiveness and broad applications.

2. Related Work

2.1. Vector Font Generation

In early years, researchers tried to generate glyph images and utilized traditional vectorization methods [17, 24] to obtain vector fonts. With the development of sequential models such as long short-term memory RNNs [13] and Transformer [31], a number of methods have been developed that treat vector graphics as drawing-command sequences for modeling. Lopes et al. [20] proposed SVG-VAE that provides a scale-invariant representation for imagery and then uses an RNN decoder to generate vector glyphs. Carlier et al. [3] proposed a novel hierarchical generative network called DeepSVG, which effectively disentangles high-level shapes from the low-level commands that encode the shape itself and directly predicts a set of shapes in a non-autoregressive fashion. Reddy et al. [26] proposed Im2Vec that can generate complex vector graphics with varying topologies, and only requires indirect supervision from raster images. Mo et al. [22] introduced a general framework to produce line drawings from a wide variety of images by using a dynamic window. At the same time, Wang et al. [33] adopted dual-modality learning to synthesize visually pleasing vector glyphs. However, all the methods mentioned above are based on the explicit representation of vector fonts, which requires the neural networks to learn long-range dependence and generate the drawing commands step by step. On the contrary, our method regards the vector glyph as a whole and is trained by the deterministic values of signed distance functions (SDFs).

2.2. Implicit Shape Representation

In the last two decades, lots of distance functions-based algorithms have been developed for implicit shape representation. As a pioneering work, Gibson et al. [8] proposed an adaptively sampled distance field using octree for both 2D glyph representation and 3D rendering. Green [9] used distance functions for glyph rasterization. With the rapid development of machine learning techniques, signed distance functions and other implicit functions-based methods have been widely used in 2D and 3D reconstruction tasks. Park et al. [25] proposed DeepSDF which uses an auto-decoder to predict the values of signed distance functions for 3D reconstruction. Chen et al. [5] presented an implicit function using fully-connected layers that output signed distances with the input of image features and sampling points. However, the implicit representations they used can not be directly converted to traditional explicit descriptions.

Aiming at 3D shape decomposition, Deng et al. [6] proposed CVXNet and Chen et al. [4] proposed BSP-Net, which both employ hyperplanes to split the space to get convex hulls. However, such methods only work well on simple convex objects, performing poorly on shapes with concave curves. Liu et al. [19] replaced the hyperplanes with quadratic curves. It does successfully reconstruct glyph images with an arbitrary resolution, but cannot convert its output to conventional Bézier curves. On the contrary, our method is able to reconstruct and generate visually-pleasing

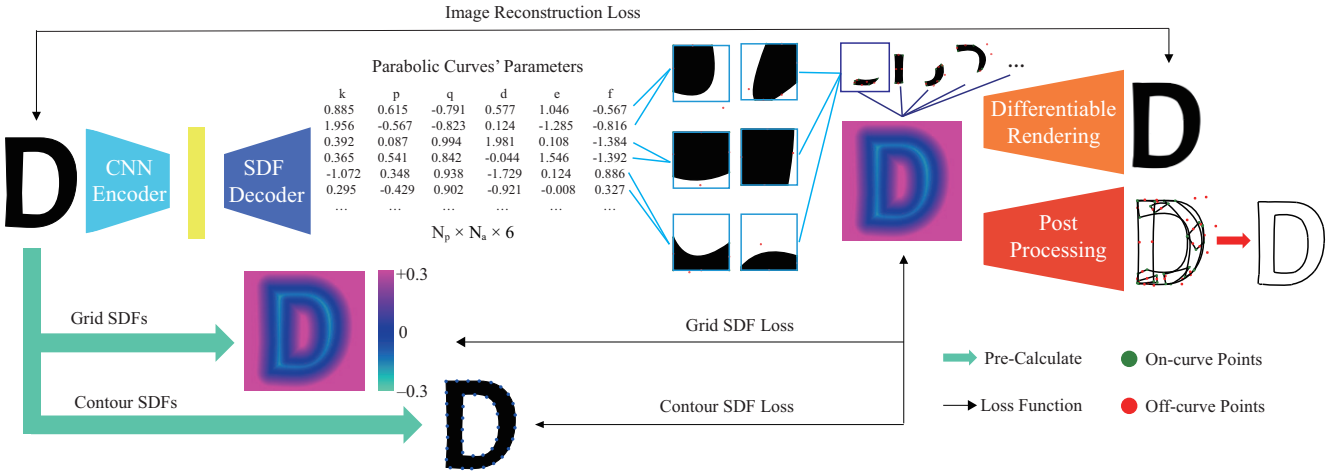


Figure 2. An overview of our vector font reconstruction framework.

vector fonts using a new explicit shape representation based on our proposed pseudo distance functions.

3. Method Description

In order to directly reconstruct and synthesize vector fonts from input glyph images via neural networks, we need a learnable shape representation that precisely preserves geometry information. Previous methods that fully rely on sequential generative models such as SVG-VAE [20] and DeepVecFont [33] fail to perfectly handle this task mainly due to the ambiguity of control point selection. To address this problem, we propose a vector graphic reconstructor based on an interpretable distance function with differentiable rendering. Unlike most SDF-based methods such as DeepSDF [29] which directly predicts the values of signed distance functions via neural networks, our model outputs the learnable parameters of curves that can be applied to handle complex glyph outlines containing concave curves.

Fig. 2 shows the pipeline of our VecFontSDF. For data preparation, we need to pre-calculate the values of signed distance functions (SDFs) for each input glyph using the method described in Sec. 3.1. We have two types of pre-calculated SDFs: grid SDFs and contour SDFs. Grid SDFs mean that the sampling points used to calculate SDFs are located at all grid positions (i.e., pixels in a raster image). Contour SDFs mean that the sampling points are uniformly distributed near the contours of the input glyph. Then, a CNN encoder is trained to extract the features from input raster images and an SDF decoder is followed to predict the parameters for every parabolic curve. We design a new pseudo distance function to calculate the distance from every sampling point to the parabolic curve based on the predicted parameters. The values of pseudo distance functions are supervised by real SDFs to achieve more precise reconstruction results. Furthermore, with a simple but effective differentiable renderer, SDFs can be converted into raster images which are compared with input glyph images. Finally, the above parameters of parabolic curves are converted into quadratic Bézier curves after the post-processing step. More details of our VecFontSDF are described in the following subsections.

3.1. Signed Distance Functions

We consider a glyph $C = \{c_1, c_2, ..., c_{N_c}\}$ to be a set of N_c contours c_i, and each c_i is defined as a sequence of Bézier curves $c_i = \{B_1, B_2, ..., B_{N_B}\}$, where N_B indicates the amount of Bézier curves in the sequence c_i. To generate practical vector font libraries (e.g., TTF fonts), our model adopts the quadratic Bézier curve B which is defined as:

$$B : P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, 0 \le t \le 1, \quad (1)$$

where P_0 and P_2 denote the first and last control points (on-curve points), respectively, and P_1 is the intermediate (off-curve) control point (see Fig. 2).

To calculate the signed distance function of a given point $P(x, y)$ towards the glyph outline, we first need to determine its sign, and then find its nearest curve $\hat{B}$ and the corresponding parameter $\hat{t}$:

$$\begin{aligned} \hat{t} &= \text{argmin}_t ||P(t) - P||_2^2 \\ ||P(t) - P||_2^2 &= (x(t) - x)^2 + (y(t) - y)^2 \triangleq f^4(t). \end{aligned} \quad (2)$$

Since the distance function is a quartic function, we get its minimum point by letting its derivative equal to zero. After obtaining $\hat{t}$, we can calculate the distance value:

$$\text{dist}^2(\hat{t}) = \left|\left|P(\hat{t}) - P\right|\right|_2^2 = (x(\hat{t}) - x)^2 + (y(\hat{t}) - y)^2, \quad (3)$$

and its sign:

$$\text{sgn}(\hat{t}) = \frac{\overrightarrow{P(\hat{t})P} \times \overrightarrow{v(\hat{t})}}{\left|\overrightarrow{P(\hat{t})P} \times \overrightarrow{v(\hat{t})}\right|}, \quad (4)$$

where $v(t)$ denotes the derivative of $P(t)$:

$$v(\hat{t}) = -2(1 - \hat{t})P_0 + 2(1 - 2\hat{t})P_1 + 2\hat{t}P_2. \quad (5)$$

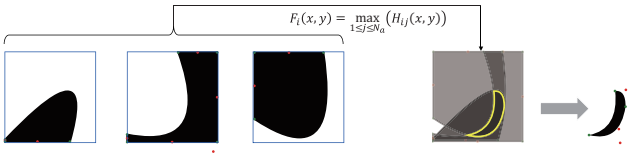


Figure 3. An illustration of calculating the intersection of N_a areas via maximum operator.

Finally, the signed distance is computed by:

$$D(x,y) = \text{dist}(\hat{t}) \times \text{sgn}(\hat{t}). \tag{6}$$

To fully exploit the glyph's vector information, we repeat the above calculation on all grid positions and uniformly sample points near glyph contours to obtain grid SDFs and contour SDFs, respectively. An illustration of the proposed grid SDFs and contour SDFs can be found in Fig. 5.

3.2. Pseudo Distance Functions

Due to the complex gradient back propagation process, directly calculating the real signed distances from sampling points to parabolic curves via Eq. 6 is infeasible when training our neural network. To solve this problem, we propose a pseudo distance function inspired by the algebraic distance [10] to simulate the original signed distance function. As mentioned above, our model aims to generate a set of parabolic curves which are defined by:

$$k(px+qy)^2 + dx + ey + f = 0. \tag{7}$$

Thus, we can define the pseudo distance function of a sampling point to a parabolic curve as:

$$H(x,y) = k(px+qy)^2 + dx + ey + f. \tag{8}$$

Similar to the original signed distance function, our pseudo distance function also regards the point with a positive distance value as outside and vice versa. The area inside a parabolic curve is defined as $H(x,y) < 0$. To reconstruct the geometry of an input glyph image (see Fig. 3), we compute the intersection of N_a areas to get a shape primitive:

$$F_i(x,y) = \max_{1\le j\le N_a}(H_{ij}(x,y)) \quad \begin{cases} > 0 & \text{outside} \\ \le 0 & \text{inside} \end{cases}, \tag{9}$$

where $F_i(x,y) < 0$ denotes the i-th primitive, and N_p denotes the number of primitives used to reconstruct the glyph image. An illustration of the above maximum operator is shown in Fig. 3.

Finally, we get the proposed pseudo distance field by:

$$G = \min_{1\le i\le N_p}(F_i) \quad \begin{cases} > 0 & \text{outside} \\ \le 0 & \text{inside} \end{cases}, \tag{10}$$

where $G_i < 0$ denotes the combination of N_p primitives. Thus, our model outputs $N_p \times N_a$ parabolic curves where each curve has 6 parameters $\{k, p, q, d, e, f\}$ as defined in Eq. 7. Our representation is able to depict the concave regions of a glyph due to the utilization of the parameter k whose sign determines whether a region is inside or outside.

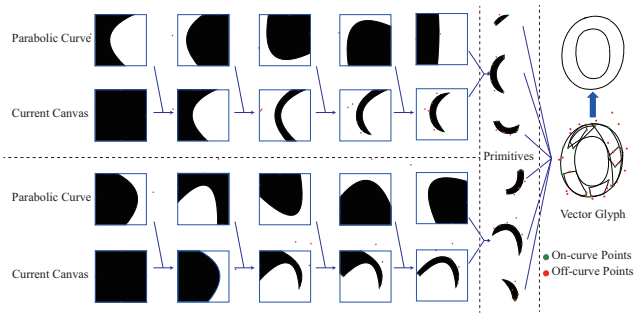


Figure 4. A demonstration of our post processing step.

3.3. End-to-End Vector Reconstruction

After obtaining the final pseudo distance field G, we need to render it to the raster image in an differentiable manner. Inspired by [27], we rasterize $G \in (-\infty, \infty)$ to the glyph image $\hat{I} \in [0,1]$ by computing:

$$\hat{I}(g_{x,y}) = \begin{cases} 1 & \gamma < g_{x,y} \\ \frac{1}{2} - \frac{1}{4}\left(\left(\frac{g_{x,y}}{\gamma}\right)^3 - 3\left(\frac{g_{x,y}}{\gamma}\right)\right) & -\gamma \le g_{x,y} \le \gamma \\ 0 & g_{x,y} < -\gamma \end{cases}, \tag{11}$$

where γ represents the learnable range of SDF values. Only the position whose pseudo distance value belongs to $[-\gamma, \gamma]$ can back-propagate its gradient to update the network.

Our model is trained by minimizing the loss function that consists of the image reconstruction loss L_{image}, the grid SDF loss L_{grid}, the contour SDF loss $L_{contour}$, and the regularization loss $L_{regular}$. The image reconstruction loss is defined as the mean square error between the reconstructed image $\hat{I}$ and the input image I:

$$L_{image} = \left|\left|\hat{I} - I\right|\right|_2^2. \tag{12}$$

To further exploit the vector information, we use the precalculated grid SDFs and contour SDFs to provide a much more precise supervision. However, due to the inconsistency of calculation between the real distance field D from Eq. 6 and the pseudo distance field G from Eq. 10, we cannot directly calculate their numerical difference. But since they share the same zero level-set and monotony with the movement of sampling points, we only calculate the loss when they have different signs, which means the predicted curve and the ground-truth curve are on the opposite side of a sampling point. The contour SDF loss is defined as:

$$L_{contour} = \frac{1}{M_c} \sum_{x,y \in S_{contour}} \text{ReLU}\left(-G(x,y) \times D(x,y)\right), \tag{13}$$

where $S_{contour}$ denotes the set containing sampling points uniformly distributed near the glyph contours as described in Sec. 3.1, and $M_c = |S_{contour}|$.

Figure 5. Examples of vector glyphs, glyph images, grid SDFs and contour SDFs. For better visualization of grid SDFs, we map the positive value of distance functions to the red channel of RGB images and map the negative value to the green channel. To visualize the contour SDFs, we add all the contour sampling points into the original SVG files. Please zoom in for better inspection.

Similarly, the grid SDF loss L_{grid} can be computed by:

$$L_{grid} = \frac{1}{M_g} \sum_{x,y \in S_{grid}} \text{ReLU}\left(-G(x,y) \times D(x,y)\right), \tag{14}$$

where S_{grid} denotes the set consisting of $M_g = W \times H$ points sampled from all pixels of input image with the width W and height H.

To further improve the quality of reconstructed images, we also introduce the regularization loss $L_{regular}$ to our model. Specifically, we first need to normalize the parameters p and q used in Eq. 8 by forcing $p^2 + q^2 = 1$. Then, we restrict the minimum value of $\hat{k}^2$ outputted by our model greater than a pre-defined k^2 to obtain a clear image. The regularization loss $L_{regular}$ is defined as:

$$\begin{aligned} L_{regular} = \frac{1}{N_p \times N_a} \Big(\lambda_{k^2} \sum \text{ReLU}\left(k^2 - \hat{k}^2\right) \\ + \sum \left(p^2 + q^2 - 1\right)^2 \Big). \end{aligned} \tag{15}$$

Finally, the complete loss function of our model is defined as the weighted sum of all above losses:

$$\begin{aligned} L_{total} = \lambda_{image} L_{image} + \lambda_{grid} L_{grid} \\ + \lambda_{contour} L_{contour} + \lambda_{regular} L_{regular}. \end{aligned} \tag{16}$$

3.4. Post Processing

In order to synthesize vector fonts for practical uses, we need to convert the parameters of parabolic curves generated by our SDF decoder to quadratic Bézier curves. Fig. 4 illustrates the post-processing step of our method. For each primitive, our initial canvas starts from a square and each curve splits the space into an inside region and an outside region. Then, we calculate the intersection of the inside region of newly-added curve and the current canvas to update the canvas recursively to get the final primitive. Finally, we assemble all the primitives together to form the output vector glyph. What's more, we further merge the outlines of all primitives to get a complete outline representation of the glyph. More details regarding how to calculate the control points of quadratic Bézier curves in those primitives and merge their outlines are shown in supplementary materials.

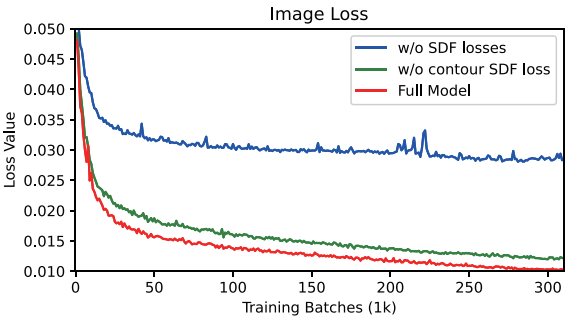


Figure 6. The loss curves of our model evaluated on the test set when training with different losses.

4. Experiments

We conduct both qualitative and quantitative experiments on the dataset released by [23], which consists of 1,116 fonts. We use the first 1,000 fonts for training and the rest 116 for testing. We use the method described in Sec. 3 to convert the SVG file of each vector glyph to grid SDFs and contour SDFs. Meanwhile, we use the traditional rasterization method to render every vector glyph to the corresponding glyph image with the resolution of 128×128. Examples of vector glyphs, glyph images, grid SDFs and contour SDFs are shown in Fig. 5.

In our experiments, all the input and output images are in the resolution of 128×128. We calculate the grid SDFs on the center position of every pixel, i.e., $(0.5, 0.5), (0.5, 1.5), (0.5, 2.5), ..., (127.5, 127.5)$ under our image resolution setting (i.e., $M_g = 128 \times 128$). The total number of contour SDFs sampling points M_c are set to 4,000. Locations of all the sampling points are normalized to [-1,1] during training for better stability. After normalization, the threshold γ in Eq. 11 is set to 0.02. The encoder to extract image feature used in our VecFontSDF is ResNet-18 [12] with Leaky Relu [21] and Batch Normalization [14]. For the implicit SDF decoder, we set $N_p = 16$ and $N_a = 6$ for each primitive. Weights in the loss function are selected as $\lambda_{image} = 1$, $\lambda_{grid} = 100$, $\lambda_{contour} = 1000$, $\lambda_{regular} = 1$ and $\lambda_{k^2} = 0.1$. We utilize Adam [16] with $lr = 1e-4$ and $betas = (0.9, 0.999)$ as the optimizer. We train our vector glyph reconstruction model with the batch size 64 for 100,000 iterations.

Table 1. Quantitative results of our VecFontSDF using different losses evaluated on the test set for vector font reconstruction.

Method	L_1 distance ↓	IoU↑	PSNR↑	LPIPS↓	SSIM↑
w/o SDF losses	0.0445	0.9569	17.7928	0.2327	0.8620
w/o contour SDF loss	0.0110	0.9876	25.9785	0.0384	0.9587
Full Model	**0.0090**	**0.9901**	**27.8502**	**0.0303**	**0.9669**

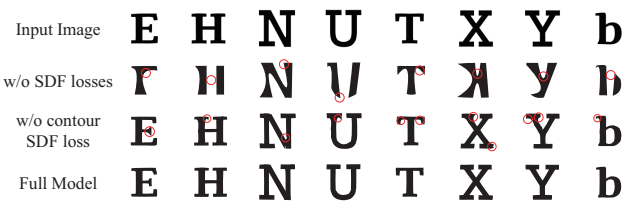


Figure 7. Qualitative results of our model using different losses. Red circles highlight the shortcomings of VecFontSDF variants.

4.1. Ablation Study

For the purpose of analyzing the impacts of different losses, we conduct a series of ablation studies by removing the corresponding loss terms used in our model on the vector reconstruction task. The loss curves of L_{image} on the test set shown in Fig. 6 demonstrate the effects of our proposed SDF losses. From the image loss curves, we witness a notable improvement brought by introducing the gird SDF loss. It's obvious that the grid SDF loss provides a much stronger supervision of the values of the output SDFs on every grid position. The raster images only contain the pixel value on every grid position where most of them are zeros or ones. As a consequence, using only the raster images to guide the training of networks losses a great deal of information. Moreover, introducing the contour SDF loss further improves the model's ability in the inference stage.

For further evaluation, we also apply several commonly-used metrics for image synthesis: L_1 distance, Intersection over Union (IoU), Peak Signal to Noise Ratio (PSNR), Learned Perceptual Image Patch Similarity (LPIPS) [35] and structural similarity (SSIM) [34] distance between the reconstructed images rendered by output SDFs and input glyph images. Table 1 shows that using all the losses achieves the best performance on the task of vector font reconstruction.

Fig. 7 provides some qualitative results to verify the effectiveness of each proposed loss. Simply using the image loss, our model even fails to synthesize the correct glyph shape since the raster image only provides the sign of distance functions on every grid point. Using the grid SDF loss results in a far more accurate supervision in leading the model to capture the geometry information of input images. But the model is still incapable of handling the details of glyph contours, e.g., the areas in red circles in Fig. 7. The contour SDF loss successfully solves this problem and helps our model sufficiently learn these details of glyph contours, such as corners and serifs.

Table 2. The comparison of quantitative results obtained by our method, BSP-Net [4] and IGSR [19] on the reconstruction task.

Method	L_1 distance ↓	IoU↑	PSNR↑	LPIPS↓	SSIM↑
BSP-Net [4]	0.0194	0.9854	22.4926	0.0500	0.9335
IGSR [19]	0.0161	0.9837	24.6895	0.0304	0.9478
VecFontSDF (64×64)	**0.0120**	**0.9866**	**28.6739**	**0.0174**	**0.9662**

4.2. Vector Font Reconstruction

Fig. 8 shows the vector font reconstruction results of our method on the test set and the corresponding glyph contours after post processing, from which we can see that almost all vector glyphs reconstructed by our VecFontSDF are approximately identical to the corresponding input glyph images. Furthermore, our reconstructed vector fonts have the same visual effects as the human-designed vector fonts, without the loss of smoothness and shape deformation.

In Fig. 9, we compare our method with two other existing SDF representations [4, 19]. Since the original image resolution used in these two methods is 64×64, for fair comparison, we also resize the resolution of our input images and the size of sampling grid to 64×64. From Fig. 9 we can see that BSP-Net [4] only uses straight hyperplanes which performs poorly on glyph images containing a large number of Bézier curves. IGSR [19] uses the general conic section equation $ax^2 + bxy + cy^2 + dx + ey + f = 0$ to depict every curve which can not be precisely converted to the quadratic Bézier curve. What's more, both of them are only supervised by the L_2 image loss, leading to blur and imprecise reconstruction results.

Table 2 shows the quantitative results of different methods calculated using the L_1 distance, IoU, PSNR, LPIPS and SSIM, further demonstrating the effectiveness and superiority of our VecFontSDF to the state of the art.

We also compare our method with two recently-proposed vector reconstruction approaches: Im2Vec [26] and multi-implicits [27] in Fig. 10. Im2Vec [26] prefers to first fit the big outlines but sometimes ignores the small ones in glyphs (e.g., "B" in column 1). It also tends to be stuck in local optimums for glyphs with multiple concave regions (e.g., "M" in column 4). Multi-implicits [27] results in finer details than Im2Vec but both of them exhibit unsmooth edges and rounded corners. What's more, Im2Vec produces hundreds of cubic Bézier curve control points to compose the glyph contour while most of them are redundant. Multi-implicits even needs to find thousands of points on the zero level-set of 2D SDF to vectorize the output. Thus, both of them are not suitable to generate practical vector fonts. On the contrary, glyphs synthesized by our method only contain dozens of quadratic Bézier curves which are more similar to human-designed ones. Detailed vector results containing all the control points are shown in the last column of Fig. 10, where green points denote on-curve control points and red points denote off-curve control points.

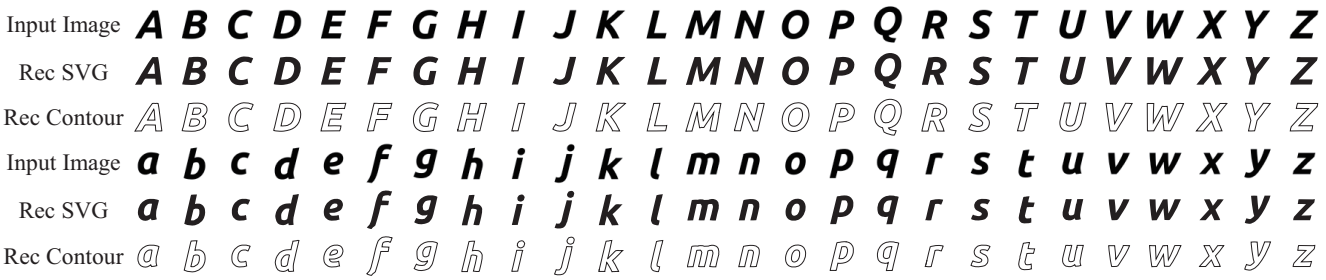


Figure 8. Examples of our vector font reconstruction results on the test set. "Rec SVG" denotes the filled shapes of the reconstructed vector glyphs and "Rec Contour" denotes the contours of corresponding vector glyphs. Please zoom in for better inspection.

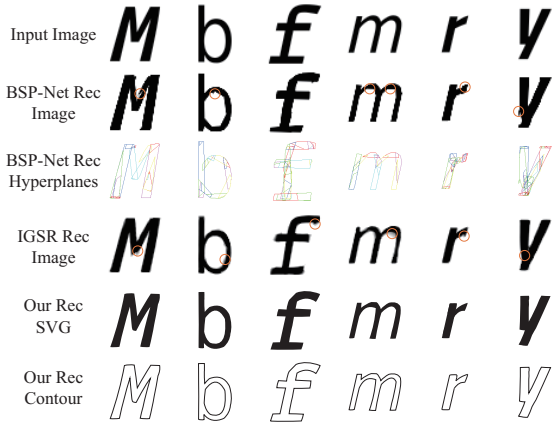


Figure 9. The qualitative comparison of our method and other SDF-based methods [4, 19]. "Rec" denotes "Reconstructed". Orange circles highlight the poor performance of previous methods.

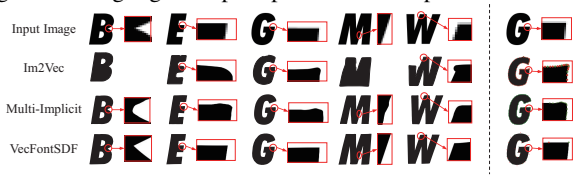


Figure 10. Vector reconstruction results of Im2Vec [26], multi-implicits [27] and our method. Please zoom in for more details.

4.3. Vector Font Interpolation

We also conduct experiments on vector glyph interpolation to further prove the ability of our representation. Given two input glyph images of the same character, we use the pre-trained image encoder on vector font reconstruction to extract the corresponding latent codes z_1, z_2. Then, an interpolated feature of them can be computed by $z = (1 - \lambda) \cdot z_1 + \lambda \cdot z_2$.

We feed the latent code z into our pre-trained SDF decoder and obtain the interpolated vector glyphs after post processing. Fig. 11 shows that our model achieves smooth interpolation between different styles of fonts and is capable of generating visually-pleasing new vector fonts by only inputting raster glyph images.

4.4. Few-Shot Style Transfer

To further demonstrate the potential of our VecFontSDF for vector font generation, we directly extend the popular

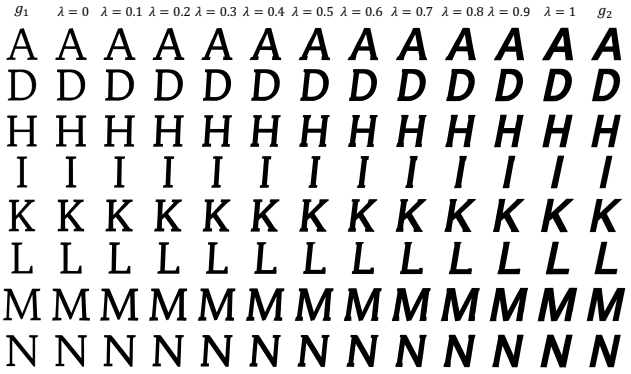


Figure 11. Examples of vector font interpolation results obtained by VecFontSDF. Our model can generate a series of high-quality vector fonts in new styles by only providing raster glyph images in two different font styles (the first and last columns).

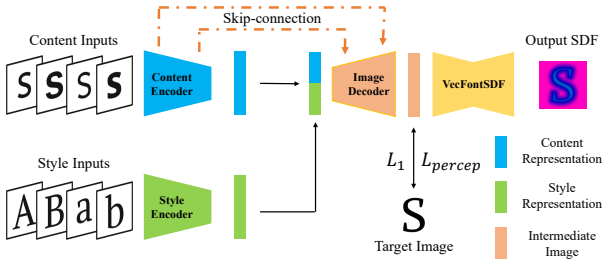


Figure 12. The pipeline of our few-shot style transfer model.

few-shot style transfer task from the image domain to the vector graphic domain. The input of our few-shot style transfer model consists of the style reference set $\mathcal{R}_{S_i}$ and the content reference set $\mathcal{R}_{C_i}$. Our goal is to output the vector glyph which has the same style as $\mathcal{R}_{S_i}$ and the same content as $\mathcal{R}_{C_i}$. The network architecture is shown in Fig. 12.

First, we use two separated CNN encoders to extract the style feature z_s of $\mathcal{R}_{S_i}$ and the content feature z_c of $\mathcal{R}_{C_i}$. Then we simply concatenate z_s and z_c and send it into the image decoder to get an intermediate image $\hat{X}_c$. We add skip connections between the content encoder and the image decoder like U-net [28]. Our pretrained VecFontSDF obtained on the vector reconstruction task is followed to process the intermediate image and output corresponding SDFs. The intermediate image is supervised by L_m which consists of the L_1 loss and the perceptual loss [15] com-

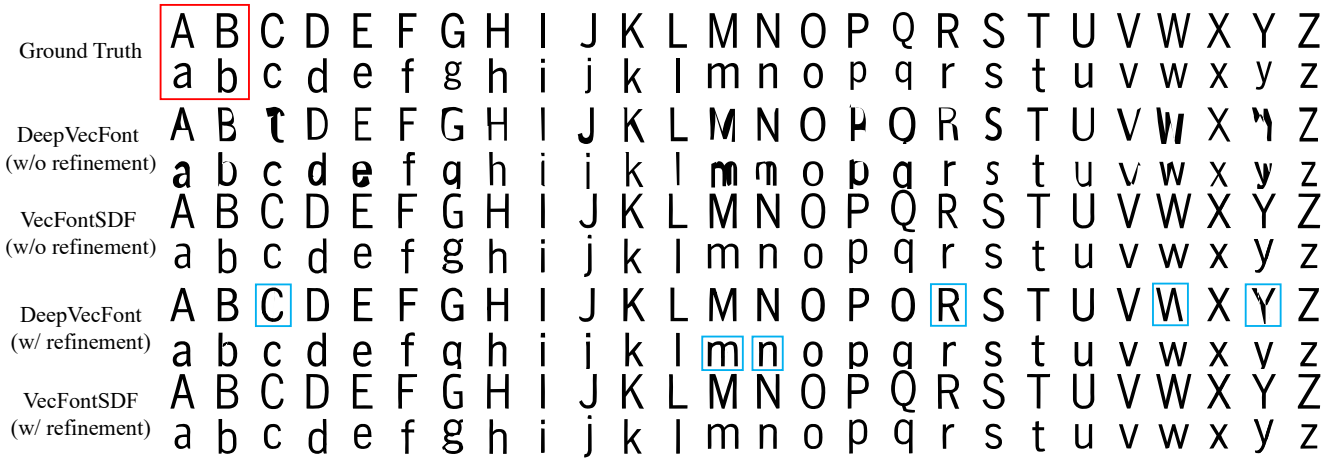


Figure 13. Few-shot vector font generation results of our VecFontSDF and a state-of-the-art method DeepVecFont [33] before and after refinement. The input reference glyphs are marked by red rectangles. Note that our model only needs the four glyph images as input while DeepVecFont also needs these vector glyphs. Please zoom in for better inspection.

Figure 14. Typical failure cases of our method when handling glyphs with irregular shapes. "Rec" denotes "Reconstructed".

pared with the groud-truth image X_c. The objective function of our model is the sum of L_m and L_{total} which is defined in Eq. 16. Our few-shot style transfer model is end-to-end trainable since the proposed VecFontSDF is differentiable. The two CNN encoders and image decoder use the residual block [12] with layer normalization [2], and the same optimizer configuration as mentioned in Sec. 4 is adopted.

In Fig. 13, we compare the performance of our few-shot style transfer model with DeepVecFont [33] on our test set. DeepVecFont receives both raster images and vector graphics as input while our model only needs raster images. Moreover, our model is an end-to-end framework in the reference stage without offline refinement. Therefore, we compare our synthesized vector glyphs with the results of DeepVecFont before and after refinement. From Fig. 13 we can see that the advantage of the explicit representation of DeepVecFont is that it outputs relatively smooth glyph contours. However, due to the difficulty of handling the long-range dependence, DeepVecFont faces serious shape distortion problems. On the contrary, our method simultaneously outputs all the parameters of parabolic curves which effectively avoids this issue. Although Diffvg [18] has a strong ability to align the generated vector glyphs with the target images, the refinement performance relies heavily on the quality of input vector glyphs. Due to the severe distortions and artifacts in some synthesized initial glyphs, there still exist many failure cases of DeepVecFont, where some thin lines and even artifacts may appear after refinement (glyphs marked in blue boxes and it would be better to open with Adobe Acrobat to see these thin lines). It can be concluded that our few-shot vector font synthesis method based on VecFontSDF markedly outperforms the state of the art, indicating the broad application of VecFontSDF.

4.5. Failure Cases

Although VecFontSDF enables high-quality vector font reconstruction under most circumstances, it still has difficulty to model complex glyph shapes occasionally. As shown in Fig. 14, our method synthesizes inaccurate vector glyphs for some strange shapes, like "G" and "S". It also faces degradation on some cursive glyphs like "f", "M" and "q". What's more, it is also a tough task for VecFontSDF if input glyph images have a lot of disconnected regions like "N" in the last column. This is mainly because our model struggles to cover such complicated geometries using a limited number of shape primitives.

5. Conclusion

In this paper, we presented a novel vector font shape representation, VecFontSDF, which models glyphs as shape primitives enclosed by a set of parabolic curves that can be translated to commonly-used quadratic Bézier curves. Experiments on vector font reconstruction and interpolation tasks verified that our VecFontSDF is capable of handling concave curves and synthesizing visually-pleasing vector fonts. Furthermore, experiments on few-shot style transfer demonstrated the ability of our VecFontSDF for many generation tasks. In the future, we are planning to upgrade our model by utilizing more effective network architectures to address the above-mentioned problems of our model when handling glyphs with irregular shapes, and extend our Pseudo Distance Functions to higher-order curves.

References

[1] 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. 1

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 8

[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, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 6, 7

[5] 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. 2

[6] 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. Computer Vision Foundation / IEEE, 2020. 2

[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. 1

[8] Sarah F. Frisken Gibson, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In Judith R. Brown and Kurt Akeley, editors, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, pages 249–254. ACM, 2000. 2

[9] Chris Green. Improved alpha-tested magnification for vector textures and special effects. In Sara McMains and Peter-Pike Sloan, editors, *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2007, San Diego, California, USA, August 5-9, 2007, Courses*, pages 9–18. ACM, 2007. 2

[10] Gaël Guennebaud and Markus H. Gross. Algebraic point set surfaces. *ACM Trans. Graph.*, 26(3):23, 2007. 4

[11] David Ha and Douglas Eck. A neural representation of sketch drawings. *international conference on learning representations*, 2017. 1

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5, 8

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 5

[15] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016. 7

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[17] Alexander Kolesnikov and Pasi Fränti. Polygonal approximation of closed discrete curves. *Pattern Recognition*, 40(4):1282–1293, 2007. 2

[18] 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, 8

[19] 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, 6, 7

[20] 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. 1, 2, 3

[21] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013. 5

[22] Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. General virtual sketching framework for vector line art. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021. 2

[23] Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics (TOG)*, 33(4):1–9, 2014. 5

[24] Wanqiong Pan, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Skeleton-guided vectorization of chinese calligraphy images. In *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE, 2014. 2

[25] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 165–174. Computer Vision Foundation / IEEE, 2019. 1, 2

[26] 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. 2, 6, 7

[27] Pradyumna Reddy, Zhifei Zhang, Zhaowen Wang, Matthew Fisher, Hailin Jin, and Niloy Mitra. A multi-implicit neural representation for fonts. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12637–12647. Curran Associates, Inc., 2021. 4, 6, 7

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 7

[29] Dmitriy Smirnov, Matthew Fisher, Vladimir G Kim, Richard Zhang, and Justin Solomon. Deep parametric shape predictions using distance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 561–570, 2020. 3

[30] Shusen Tang, Zeqing Xia, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Fontrnn: Generating large-scale chinese fonts via recurrent neural network. In *Computer Graphics Forum*, volume 38, pages 567–577. Wiley Online Library, 2019. 1

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 2

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

[33] 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, 3, 8

[34] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 6

[35] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6