QT-Font: High-efficiency Font Synthesis via Quadtree-based Diffusion Models

Yitian Liu
lsflyt@pku.edu.cn
Wangxuan Institute of Computer Technology
Peking University
Beijing, China

Zhouhui Lian*
lianzhouhui@pku.edu.cn
Wangxuan Institute of Computer Technology
Peking University
Beijing, China

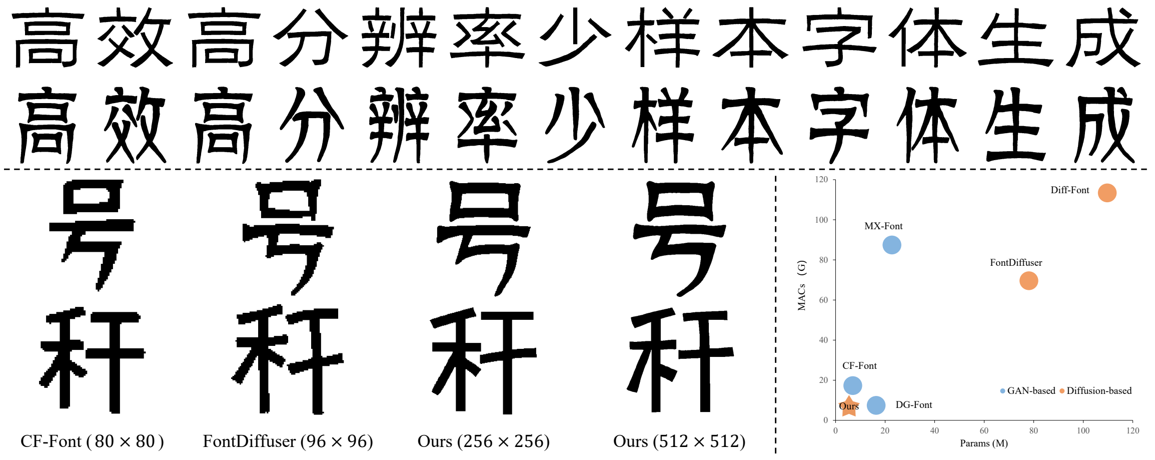

Figure 1: Examples of glyph images synthesized by different models are shown in the top part and the center of the figure, which illustrates that high-resolution glyph images exhibit smooth edges and superior details. On the contrary, low-resolution glyph images have jagged edges at the same size, significantly impairing the aesthetic appeal. This paper introduces QT-Font, an effective model for generating high-resolution glyph images. As shown on the right bottom, our QT-Font significantly reduces both the number of parameters and computational load compared to other methods, especially diffusion-based models.

ABSTRACT

Few-shot font generation (FFG) aims to streamline the manual aspects of the font design process. Existing models are capable of generating glyph images in the same style of a few input reference glyphs. However, mainly due to their inefficient glyph representations, these existing FFG methods are limited to generating low-resolution glyph images. To address this problem, we introduce QT-Font, an efficient quadtree-based diffusion model specifically designed for FFG. More specifically, we design a sparse quadtree-based glyph representation to reduce the complexity of the representation space, exhibiting linear complexity and uniqueness. Concurrently, to reduce computational complexity, we propose a U-net model based on the dual quadtree graph network and the discrete diffusion model. Furthermore, a content-aware pooling module is also adopted to lessen the computational demands of the diffusion process. Qualitative and quantitative experiments have been conducted to demonstrate that our QT-Font, compared to existing approaches, can generate high-resolution glyph images with superior quality and more visually pleasing details, meanwhile significantly reducing both parameter sizes and computational costs.

CCS CONCEPTS

• **Computing methodologies → Image representations**.

KEYWORDS

Image synthesis, Font synthesis, Discrete diffusion model

ACM Reference Format:
Yitian Liu and Zhouhui Lian. 2024. QT-Font: High-efficiency Font Synthesis via Quadtree-based Diffusion Models. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27–August 01, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3641519.3657451

*Zhouhui Lian is the corresponding author.

1 INTRODUCTION

Fonts are widely used in our daily life. However, manual font design is a time-consuming task, and the quality of the resulting font heavily relies on the designer's experience. To meet the increasing demand for fonts, the challenging task of few-shot font generation (FFG) has become necessary. The aim of the FFG task is to generate high-quality fonts using only a few reference glyphs, significantly improving the efficiency of font design. However, existing FFG methods based on convolutional neural networks (CNNs) can only generate low-resolution glyph images, such as DG-Font (80×80) [Xie et al. 2021], CF-Font (80×80) [Wang et al. 2023], MX-Font (128×128) [Park et al. 2021]. This is mainly due to the high representation complexity of the rasterized image, typically $O(L^2)$ where L denotes the image resolution. Therefore, these models require substantial computational resources to process high-resolution glyph images, making it challenging to guarantee their performance at high resolutions. On the other hand, some researchers attempt to use resolution-independent representations to directly generate vector glyphs that can be arbitrarily scaled without loss of detail, including DeepVecFont [Wang and Lian 2021] and VecFontSDF [Xia et al. 2023]. These methods use vector outlines [Aoki and Aizawa 2022; Lian and Gao 2022] or implicit shape representations [Chen et al. 2023; Reddy et al. 2021] to represent glyphs. Compared to the former, these representations are sparser and more efficient, but they are uncertain. For example, a Bézier curve can be divided into two or more curves that collectively represent the same shape. The uncertainty makes it difficult for these methods to deal with complex glyphs.

To generate high-quality and high-resolution glyphs, a sparse glyph representation is essential. This representation should possess the following advantages: 1) low complexity, with a complexity lower than $O(L^2)$ and approaching linear complexity; 2) uniqueness, ensuring a bijective mapping between the glyph and the representation. To achieve this goal, we design a quadtree glyph representation and further propose QT-Font, an efficient few-shot font generation method based on dual quadtree graph networks. Specifically, we first transform the rasterized glyph outline into a sparse point cloud to reduce **representation (space) complexity**. Subsequently, for lower **computational (time) complexity**, we represent the point cloud as a quadtree and feed it into our model. Finally, we apply discrete diffusion to enhance our model's performance while introducing the content-aware pooling module to mitigate increased computation load caused by noise.

To validate the effectiveness of our proposed QT-Font, we conduct a thorough comparison with state-of-the-art GAN-based and diffusion-based methods in the FFG task. Our extensive experimental results demonstrate that QT-Font outperforms the state of the art across all evaluation metrics while utilizing fewer parameters and computation load. Furthermore, we validate the model's generalization ability by showcasing its performance in the cross-language generation task. These experiments verify the superiority of our approach and highlight its potential for practical applications.

To sum up, major contributions of this paper are as follows:

- We apply quadtree to the FFG task for the first time and design an efficient glyph representation, which has the advantages of unique representation and lower representation complexity compared to other existing approaches.
- Based on the above glyph representation, we proposed QT-Font, a high-efficiency few-shot font generation network with low computational complexity which can rapidly generate high-resolution and high-quality glyph images. Meanwhile, we propose the content-aware pooling module to mitigate the additional computational burden caused by noise in discrete diffusion.
- Quantitative and qualitative experiments have been conducted to verify the superiority of our QT-Font and the effectiveness of the proposed modules.

2 RELATED WORK

2.1 Font Generation

2.1.1 Glyph Image Generation. Earlier methods employed GAN [Gao et al. 2019; Jiang et al. 2017; Tian 2017] or VAE [Lyu et al. 2017; Zhang et al. 2018] to transfer font styles. For example, based on MC-GAN [Azadi et al. 2018], Gao et al. [2019] designed AGIS-Net generating both glyph shape and texture with three discriminators. Zhang et al. [2018] proposed an end-to-end method for synthesizing glyph images based on U-net structure. To enhance glyph details, recent approaches have incorporated the prior knowledge of characters into their models. For instance, SA-VAE [Sun et al. 2017], CalliGAN [Wu et al. 2020], StrokeGAN [Zeng et al. 2021], and DiffFont [He et al. 2022] introduced stroke or component encoding vectors into their models. Additionally, RD-GAN [Huang et al. 2020] and Kong et al. [2022] designed component-aware discriminators. Moreover, some methods reused the features of identical components, such as LF-Font [Park et al. 2020], MX-Font [Park et al. 2021], and FS-Font [Tang et al. 2022]. Up to now, glyph image generation approaches can synthesize high-quality glyphs, particularly diffusion-based models like Diff-Font [He et al. 2022] and FontDiffuser [Yang et al. 2023]. However, due to the high representation and computational complexity, the existing methods can only generate low-resolution glyph images.

2.1.2 Vector Glyph Generation. In the early years, researchers tried vectorizing rasterized images to get the vector glyphs. For instance, SVG-VAE [Lopes et al. 2019] employed VAE mapping glyph images into features and generated vector glyphs via RNN or LSTM. However, the low-resolution of the images may result in inadequate representation of vector details. Consequently, recent studies have focused on obtaining vector shapes directly through sequential models. Carlier et al. [2020] designed DeepSVG based on Transformer, which exhibits satisfactory performance for simple graphics but demonstrates limited effectiveness when dealing with complex glyphs. For Latin vector glyphs, Wang and Lian [2021] proposed DeepVecFont, a dual-modal learning approach that combines vector synthesis with image synthesis. For Chinese characters, Aoki and Aizawa [2022] introduced a Transformer-based network. However, the above methods perform poorly in handling Chinese characters, especially for complex glyphs.

2.1.3 Implicit glyph representation. The implicit glyph representation describes the glyph through implicit geometry [Green 2007]. With the help of deep learning, neural networks can extract features automatically and construct the implicit representation. For instance, DeepSDF [Park et al. 2019] and IM-Net [Chen and Zhang 2019] utilized autoencoders and multi-layer perceptrons, respectively, to predict the signed distance field (SDF) of shapes. Subsequent studies focused on implicit representations of 3D objects such as BSP-Net [Chen et al. 2020], Cvx-Net [Deng et al. 2020], and so on. In order to describe complex corners in glyphs, Reddy et al. [2021] proposed multi-implicit neural representation, using multiple distance fields to obtain sharp corners and smooth edges. Similarly, based on BSP-Net [Chen et al. 2020], Liu et al. [2022] employed quadratic Bézier curves instead of hyperplanes to reconstruct glyphs at arbitrary resolutions. Nevertheless, obtaining the vector glyph through the implicit representation is still complicated. To tackle this issue, VecFontSDF [Xia et al. 2023] constructed the glyph by utilizing the pseudo-distance function and subsequently transformed it into vector glyphs using quadratic Bézier curves. Implicit glyph representation is more efficient compared to the rasterized glyph. But this representation is usually uncertain, which affects the performance of the generative model.

2.2 Diffusion Models

In recent years, diffusion models [Ho et al. 2020] have demonstrated remarkable performance, especially for image generation tasks [Dhariwal and Nichol 2021; Rombach et al. 2022]. These advancements have also extended to font generation tasks, such as Diff-Font [He et al. 2022], FontDiffuser [Yang et al. 2023], and Calli-Paint [Liao et al. 2023], which leverage the diffusion-based backbone and utilize specialized modules for glyph synthesis.

Most of the existing diffusion-based generation methods focus on handling continuous data. For discrete data, D3PM [Austin et al. 2021] used discretized Gaussian kernels (or absorbing state kernels) instead of Gaussian noise in the forward and backward processes. Latter, VQ-Diffusion [Gu et al. 2022] replaced the auto-regressive model in VQ-VAE with the discrete diffusion model to achieve an enhanced balance between quality and speed. However, these models are limited to generating low-resolution images [Austin et al. 2021], or their performance heavily relies on the auto-encoder [Gu et al. 2022]. To address this problem, we propose a few-shot font synthesis method based on quadtree-based diffusion models that enables efficient end-to-end generation of high-resolution glyph images.

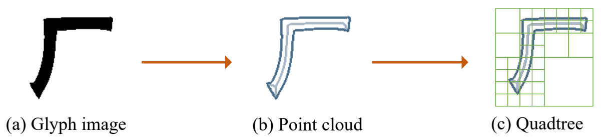

Figure 2: We extract the outline and skeleton point cloud of the glyph and represent them using a quadtree. For visual clarity, the outline and skeleton are rendered wider (in our experiments, their width is one pixel).

Table 1: Comparison of commonly used glyph representations, where L denotes the image resolution, and "Uniqueness" denotes that the representation corresponds one-to-one with the glyph.

Representation	Sparsity	Uniqueness	Complexity
Image	×	✓	$O(L^2)$
SDF	×	✓	$O(L^2)$
Vector	✓	×	$O(1)$
Implicit shape	✓	×	$O(1)$
Point cloud	✓	✓	$O(L)$
Quadtree	✓	✓	$O(L)$

3 QUADTREE GLYPH REPRESENTATION

3.1 Sparse Glyph Representation

As demonstrated in Table 1, the efficiency of existing font generation methods is primarily influenced by their glyph representations. To remove the redundancy in the binary glyph images, as shown in Figure 2, we use the outline point cloud to represent the shape of a glyph, and the skeleton point cloud to differentiate between the glyph shape and the background. Generally, point cloud generation methods necessitate the alignment of point clouds of varying sizes to a uniform size, which introduces unnecessary computations [Achlioptas et al. 2018; Nichol et al. 2022; Yang et al. 2019].

Therefore, we choose to represent the above-mentioned point clouds as a quadtree using the building algorithm proposed in O-CNN [Wang et al. 2017]. Specifically, for each point, we calculate the shuffle key [Wilhelms and Van Gelder 1992] and merge the tree-nodes from bottom to top to construct the quadtree T_q. For example, the right-bottom pixels of the glyph image in Fig 2 are all white (background pixels), so they are merged a bigger node in the quadtree. This process is parallel and has a time complexity of $O(LlogL)$, where L denotes the image resolution.

3.2 Representation Complexity Analysis

Considering that the length of the glyph outline (or the number of points in the outline point cloud) is proportional to the image resolution, we assume the complexity of our glyph representation as $O(kL)$. In our experiments, k is approximately 7.56 for 128×128 images[1], which is significantly smaller compared to the resolution L. The representation (space) complexity of the quadtree is the sum of the complexity of each layer, which can be calculated as:

$$Complexity_{Quadtree} = O(kL) + O(k\frac{L}{2}) + O(k\frac{L}{4}) + \ldots = O(2kL). \tag{1}$$

Compared to the rasterized glyph image ($O(L^2)$), the quadtree is obviously sparser and more efficient, especially at high resolutions.

3.3 Outline Filling

Due to the potential gaps in the synthesized outline point cloud, which could lead to non-watertight outlines, we develop an outline filling method as illustrated in Alg. 1. Our method utilizes the initial

[1]12.65 for 256×256 images

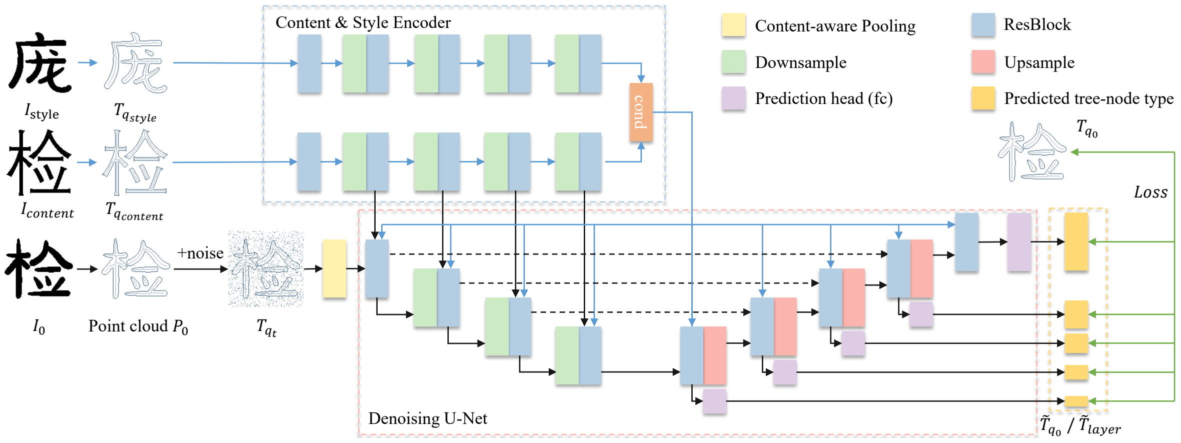

Figure 3: An overview of the proposed QT-Font, consisting a content encoder, a style encoder and a discrete denoising U-net. For linear complexity, We concatenate the condition feature into the corresponding layer of U-net.

ALGORITHM 1: Outline filling algorithm.

```
Data: The point cloud of an outline $P_o = \{(x_i, y_i)\}_{i=1}^{N_o}$; The point
      cloud of a skeleton $P_a = \{(x_i, y_i)\}_{i=1}^{N_a}$;
Result: The point cloud of a rendered image $P_i$.
$P_i = P_o + P_a$ ;
do
    $P_{new-a} = \emptyset, P_{new-o} = \emptyset, P_{Render} = \emptyset$ ;
    foreach $(x, y) \in P_a$ do
        $P_{Neighbor} = \{(x + d_x, y + d_y) | d_x, d_y \in \{-1, 0, 1\}, (d_x, d_y) \neq (0, 0)\}$ ;
        $P_{Render} = P_{Render} + P_{Neighbor}$;
        if $P_{Neighbor} \cap P_o = \emptyset$ then
            $P_{new-a} = P_{new-a} + P_{Neighbor}$;
        else
            $P_{new-o} = P_{new-o} + P_{Neighbor}$;
        end
    end
    $P_i = P_i + P_{Render}$ ;
    $P_a = P_{new-a} - P_a$ ;
    $P_o = P_{new-o} + P_o$
until $P_{new-a} = \emptyset$;
```

outline as the termination condition and expands the filled area from the skeleton point cloud. In each iteration, if the newly filled area overlaps with the outline boundary, it is added to the outline point cloud; otherwise, it is filled. This strategy effectively closes small gaps in the outline. Furthermore, due to the parallel expansion of the filling area, our algorithm maintains a time complexity of $O(L)$. As shown in Figure 4, our outline filling algorithm efficiently renders a rasterized glyph image from the synthesized point cloud.

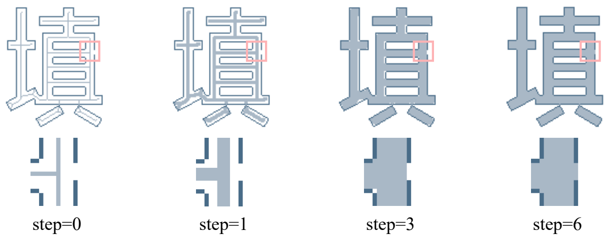

Figure 4: Through iterative outline filling, the outline and skeleton point clouds can be quickly rendered as a glyph image. The color boxes highlight the filling results of small gaps on the outline.

4 QT-FONT

4.1 Preliminary

4.1.1 Dual Quadtree Graph Networks. The quadtree convolution is similar to the graph convolution and can be formulated as follows:

$$F_i = \sum_{j \in \mathcal{N}_i} \mathcal{K}(r_{ij}) \times F_j, \tag{2}$$

where $\mathcal{K}$ is the kernel of the convolution, mapping the relationship r_{ij} between the tree-nodes t_i and t_j to the convolution weight, $\mathcal{N}_i$ denotes the neighborhood of the tree-node t_i in the quadtree T_q, and the feature of the tree-node t_i is F_i.

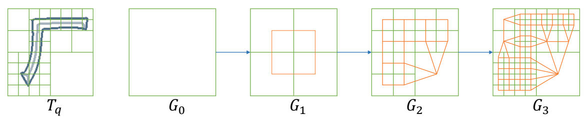

Figure 5: The top-down construction process of the dual quadtree graph.

The relationship (edge) of the quadtree consists of the Parent-Child edges (for up-sampling and down-sampling) and the Sibling edges (for convolution within the same depth). This facilitates neighborhood retrieval but limits the node receptive field. To address this limitation, Wang et al. [2022] proposed dual octree graph networks capable of computing features on nonadjacent tree-nodes. To be specific, as shown in Figure 5, the dual graph G can be sequentially constructed for each layer from top to down. In dual graphs, an octree node's neighborhood is defined by seven directions (five for quadtree): up, down, left, right, front, behind, and itself. The features of multiple tree-nodes in the same direction are merged together before the convolution (Eq. 2). This approach can expand the receptive fields of tree-nodes to enhance feature learning performance.

Computational Complexity of Dual Quadtree Graph Networks: We assume that the feature dimension is d and the number of nodes is N. The merging of tree-nodes can be executed in parallel, resulting in a time complexity of $O(5N)$, and Eq. 2 has a time complexity of $O(5Nd^2)$. Considering that the number of tree-nodes N is proportional to the resolution L, the time complexity of dual quadtree graph convolution can be denoted as $O(5Ld^2)$, which is significantly smaller than image convolution ($O(9L^2d^2)$ for a kernel size of 3×3). Consequently, by leveraging the dual quadtree graph network, our QT-Font enables rapid synthesis of high-resolution glyph images.

4.1.2 Discrete Diffusion Model. Diffusion models have been widely used in recent years, achieving impressive generative effects. Based on the sparse glyph representation mentioned in Sec 3.1, we apply the discrete diffusion model (D3PM) [Austin et al. 2021] to synthesize glyphs. D3PM includes a forward Markov process and a reverse Markov process. Unlike diffusion models defined on the continuous space, D3PM employs a transition matrix, rather than Gaussian noise, to add and remove noise from features. Assume that a scalar discrete variable x_t has K categories ($x_t \in \{0, 1, 2, ...K-1\}$), and the diffusion model contains T steps. The transition matrix $Q_t \in [0,1]^{K\times K}$ from x_{t-1} to x_t is defined as $[Q_t]_{ij} = q(x_t = i|x_{t-1} = j)$. So, the forward Markov process can be defined as:

$$q(x_t|x_{t-1}) = v(x_t)^T Q_t v(x_{t-1}), \tag{3}$$

where $v(x_t) \in \{0,1\}^K$ is a column one-hot vector of x_t. According to the Markov property, we can derive the following:

$$q(x_t|x_0) = v(x_t)^T \overline{Q}_t v(x_0),\ \overline{Q}_t = \prod_{i=1}^{t} Q_i, \tag{4}$$

$$\begin{aligned} q(x_{t-1}|x_t, x_0) &= \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \\ &= \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)} \\ &= \frac{(v(x_t)^T Q_t v(x_{t-1}))(v(x_{t-1})^T \overline{Q}_{t-1} v(x_0))}{v(x_t)^T \overline{Q}_t v(x_0)}. \end{aligned} \tag{5}$$

The reverse process of the discrete diffusion models can be defined as $p(x_{t-1}|x_t) \in [0,1]^{N\times K}$ for the N-dimensional variables x_t. Specifically, D3PM uses a neural network to predict the probabilities $\tilde{p}_\theta(x_0|x_t)$. Combining Eq. 5, we define the reverse process as:

$$p_\theta(x_{t-1}|x_t) \propto \sum_{\tilde{x}_0} q(x_{t-1}|x_t, \tilde{x}_0)\tilde{p}_\theta(\tilde{x}_0|x_t). \tag{6}$$

4.2 Discrete Diffusion Model for QT-Font

4.2.1 Details of the Network. In the proposed model, QT-Font, the point category K is set to 3, for the background, the outline, and the skeleton. Since K is small, we define Q as a uniform matrix:

$$Q_t = (1-\beta_t)I + \beta_t/K\mathbb{1}\mathbb{1}^T, \tag{7}$$

where $\beta_t \in [0,1]$ denotes a cosine schedule, and $\mathbb{1}$ is a column vector with all elements being 1.

For a quadtree with the depth D, the contribution of each tree-node at different depths to the represented glyphs is different. However, it is essential for every part of the noise to have an equal impact on the final result. Hence, we introduce noise to the point cloud instead of the quadtree. Specifically, following Eq. 4, we add noise into the point cloud P_0 to get a noisy point cloud P_t, then transform it into the quadtree T_{q_t} by using the building algorithm proposed in O-CNN [Wang et al. 2017].

As shown in Figure 3, we use a U-net network to denoise the noisy quadtree T_{q_t} with content and style references:

$$\tilde{T_{q_0}} = f(T_{q_t}, t, E_c(T_{q_{content}}), E_s(T_{q_{style}})), \tag{8}$$

where f denotes the discrete denoising U-Net, E_c and E_s represent the content encoder and the style encoder, respectively, both of which are based on dual quadtree graph networks.

To ensure the correctness of the synthesized glyphs, we feed the features of each layer in the content encoder into the U-net encoder. Moreover, to achieve linear computational complexity, we concatenate condition features with the features of the U-net, instead of the commonly-used cross-attention modules.

During inference, QT-Font samples T_{q_T} from a uniform distribution and generates T_{q_0} through $T = 20$ steps. Finally, we employ the filling algorithm proposed in Sec 3.3 to render glyph images.

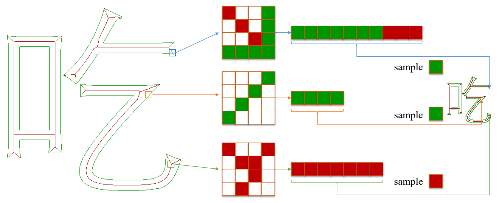

Figure 6: The architecture of the content-aware pooling module, which samples the outline and skeleton points in each patch with equal probability.

4.2.2 Content-aware Pooling Module. Since a glyph can be represented by a few outlines, we can use a quadtree to represent the glyph sparsely. However, adding noise disrupts the sparsity of the quadtree glyph representatedon, we design the Content-aware Pooling module to restrict the network computation in certain parts of the encoder. As shown in Figure 6, we split the image into patches, and sample the outline and skeleton points in the patch with equal probability. By adding the Content-aware Pooling module before the encoder of U-net, we can preserve the majority of glyph information meanwhile reducing the computational load.

4.2.3 Loss Function. The network predicts the type of each tree-node (leaf/non-leaf or background/outline/skeleton) at each layer in the decoder. Therefore, for a quadtree with a depth of D, the loss function is expressed as:

$$Loss = CrossEntropy(T_{q_0}, \tilde{T}_{q_0}) \tag{9}$$

$$= \sum_{layer=1}^{D} CrossEntropy(T_{layer}, \tilde{T}_{layer}). \tag{10}$$

Since each layer of the quadtree contributes to the glyph shape, it is difficult to give a differentiable negative variational lower bound. Therefore, we only use Eq. 10 to optimize the model (the second term of the loss function in D3PM [Austin et al. 2021]) and ignore the negative variational lower bound L_{vb}.

5 EXPERIMENTS

5.1 Experimental Settings

We collect 300 fonts and randomly divide them into the training set (270 seen fonts) and the test set (30 unseen fonts). Following VQ-Font [Pan et al. 2023], we split the 3500 glyphs into 3000 training (seen) glyphs and 500 test (unseen) glyphs. In addition, we train compared models and the proposed QT-Font using 270 fonts with 3000 glyphs (SFSC) and evaluate the performance in three test sets (SFUC, UFSC, and UFUC), respectively.

To comprehensively evaluate the superiority of our QT-Font, we employ L1 loss, FID, and LPIPS as quantitative metrics. The L1 loss measures the accuracy of synthesized glyph images at the pixel level while LPIPS and FID assess the stylistic consistency between synthesized and target glyphs. To show the generation ability across various styles, we present qualitative results for all 30 test fonts. Additionally, we also list Params (the number of parameters), MACs (the number of Multiply Accumulate) and the FPS of each model.

5.2 Implementation Details

Due to the memory limitation, we set the image resolution to 128×128, 256×256, and 512×512. In theory, QT-Font is capable of generating higher-resolution glyph images. We use AdamW to optimize the model with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and *weight decay* = 0.01. The learning rate is set to 1e-4 with cosine schedule. In the pre-training phase, the model trains for 20 epochs (about 16,000 steps) on the training set using a batch size of 1,024 (achieved by gradient accumulation). In the fine-tuning phase, we use 50 seen glyph images to fine-tune the model with a batch size of 8. The step T is set to 1,000 for training and 20 for inference, which is also applied for other diffusion-based models. For better visual effect, the style reference glyphs for each model are set to three.

5.3 Quantitative and Qualitative Comparison

We conduct a comprehensive evaluation of our proposed model QT-Font by comparing it quantitatively and qualitatively with three GAN-based FFG methods (MX-Font [Park et al. 2021], DG-Font [Xie et al. 2021], and CF-Font [Wang et al. 2023]), as well as two diffusion-based FFG methods (Diff-Font [He et al. 2022] and FontDiffuser [Yang et al. 2023]). To ensure a fair comparison, we pre-train and fine-tune each model's official implementation using our collected dataset. In cases where some models do not support high-resolution generation or increasing the resolution affects the synthesizing performance significantly, we follow the approach suggested by Fontdiffuser[2] to resize their synthesized images to match the same resolution. Moreover, Diff-Font utilizes the embedding, rather than relying on the encoder, to encode content features, and thus is limited in generating seen glyphs. So we only show the UFSC results for Diff-Font.

Quantitative results. Table 2 presents the metrics of each model on the three test sets. The proposed QT-Font demonstrates superior performance across all metrics. Notably, our FID values exhibit significant improvement compared to other state-of-the-art methods, which shows that our method can accurately mimic the target font style. Furthermore, owing to the smaller convolution kernel, the QT-Font model has a distinct advantage in the number of parameters. This reduction in parameters enhances parallelism, thereby improving synthesis efficiency while mitigating over-fitting risks.

Moreover, comparing with diffusion-based models, our method has obvious advancements in computation (MACs). By leveraging a dual quadtree graph network as the backbone, we can generate high-quality and high-resolution glyph images with less computation.

Qualitative results. Figure 8 illustrates the synthesized glyph images produced by each model on the three test sets. The glyph images generated by MX-Font exhibit some noise, negatively impacting their visual appeal. DG-Font sometimes fails to satisfactorily transfer font styles. This finding has been supported by other studies [Liu and Lian 2023; Wang et al. 2023]. CF-Font employs basis fonts to enhance style generation. However, this approach leads to poor results for rare font styles, such as those in the 16th and 20th columns (the blue boxes) of the UFSC and UFUC datasets.

Notably, diffusion-based methods yield significantly superior glyph synthesis compared to GAN-based approaches. Nevertheless, these methods still occasionally generate images with incorrect glyph details or similar but not identical font styles. In contrast to these existing techniques, our proposed model demonstrates its superior capability in generating satisfactory high-resolution glyph images while maintaining their quality. In addition, we also show our synthesis results at a resolution of 512×512 in Figure 10.

5.4 Effectiveness of Content-aware Pooling Module

Figure 7 and Table 3 present the ablation study conducted on the proposed content-aware pooling module. Random pooling disregards the importance of outline and skeleton point clouds. Therefore, the generated point cloud has significant gaps (the first five columns of Figure 7) and incorrect node types (the last three columns of Figure 7), resulting in low-quality glyph images. Conversely, removing this module from the network allows for retaining all information but incurs higher computational load. To address these issues, we introduce the content-aware pooling module to reduce computational costs while ensuring generation quality.

[2]https://github.com/yeungchenwa/FontDiffuser/issues/26

Table 2: Quantitative results of our method and other existing approaches. We evaluate each model in the resolutions of 128×128 and 256×256, respectively. P(M) denotes Params (the number of parameters), and M(G) denotes MACs (the number of Multiply Accumulate). The FPS of each model is tested on a single GTX2080Ti.

	Model	SFUC			UFSC			UFUC			P(M)↓	M(G)↓	FPS↑
		L1↓	LPIPS↓	FID↓	L1↓	LPIPS↓	FID↓	L1↓	LPIPS↓	FID↓			
128×128	MX-Font (128)	0.2230	0.2695	51.78	0.2027	0.2700	39.01	0.2052	0.2703	51.13	22.76	87.42	12.9
	DG-Font (80)	0.1801	0.2410	53.37	0.1720	0.2310	28.07	0.1734	0.2306	40.50	16.39	7.46	158.9
	CF-Font (80)	0.1288	0.1912	41.32	0.1605	0.2411	36.44	0.1624	0.2415	48.86	6.95	17.32	49.2
	Diff-Font (128)[a]	-	-	-	0.2001	0.2947	36.15	-	-	-	109.7	$113.4 \times T$	0.87
	FontDiffuser (96)[b]	0.1775	0.2356	48.37	0.1696	0.2211	24.93	0.1710	0.2206	36.28	78.05	$69.60 \times T$	1.33
	Ours (128)	**0.1281**	**0.1776**	**28.58**	**0.1209**	**0.1700**	**13.42**	**0.1286**	**0.1749**	**24.97**	4.83	$6.38 \times T$	15.6
256×256	MX-Font (128)	0.2229	0.4054	79.13	0.2025	0.3987	65.41	0.2050	0.4003	79.36	22.76	87.42	12.9
	DG-Font (80)	0.1795	0.3529	95.03	0.1717	0.3474	75.02	0.1731	0.3477	90.97	16.39	7.46	158.9
	CF-Font (80)	**0.1284**	0.2833	88.43	0.1602	0.3490	80.83	0.1622	0.3505	96.07	6.95	17.32	49.2
	Diff-Font (128)	-	-	-	0.1999	0.4109	75.08	-	-	-	109.7	$113.4 \times T$	0.87
	FontDiffuser (96)	0.1769	0.3430	74.65	0.1690	0.3309	43.47	0.1704	0.3310	57.04	78.05	$69.60 \times T$	1.33
	Ours (256)	**0.1284**	**0.2723**	**43.99**	**0.1235**	**0.2539**	**19.80**	**0.1326**	**0.2653**	**34.12**	6.20	$6.45 \times T$	13.2

[a]Diff-Font can only generate seen glyphs, so we evaluate this model with the UFSC test set.
[a]Since the training code of FontDiffuser is incomplete, we use the official pre-trained weights for evaluation.

Table 3: Ablation study results of the content-aware pooling module.

setting	SFUC L1↓	SFUC LPIPS↓	SFUC FID↓
random pooling	0.1440	0.2846	49.88
content-aware pooling	0.1361	0.2723	43.99

setting	UFSC L1↓	UFSC LPIPS↓	UFSC FID↓
random pooling	0.1257	0.2600	20.08
content-aware pooling	0.1235	0.2539	19.80

setting	UFUC L1↓	UFUC LPIPS↓	UFUC FID↓
random pooling	0.1333	0.2716	35.09
content-aware pooling	0.1326	0.2653	34.12

setting	resolution	Params(M)	MACs(G)
w/o pooling	256	6.20	6.45
w/ pooling	256	6.20	4.60
w/o pooling	512	6.52	10.07
w/ pooling	512	6.52	7.89

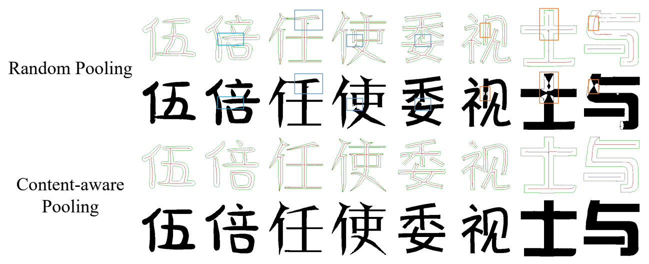

Figure 7: Comparison of synthesis results obtained by our methods with random pooling or the content-aware pooling module. Random pooling leads to large gaps in the synthesized outline and skeleton. Please zoom in to see details.

5.5 Text Rendering

To demonstrate the coherence of the style in the synthesized glyph images, we render Chinese poems in Figure 12, where glyphs synthesized by our model are highlighted in red. It is evident that QT-Font not only achieves a high level of synthesis consistency but also maintains consistent font styles with the reference glyphs.

5.6 Cross Language Synthesis

To further validate the universality of our model, we conduct cross-language style transfer by applying the learned Chinese font styles to other languages. Chinese characters typically comprise straight strokes, while Latin letters and numbers consist of numerous curved strokes. Despite these disparities, Figure 9 shows that our QT-Font can transfer font styles across a variety of languages, generating stylistically consistent results.

5.7 Limitation

Although our model is capable of generating satisfactory glyphs, there exists a limitation wherein the synthesized glyph relies on predictions from multiple layers, potentially leading to incorrect synthesis. Specifically, for the leaf outline node, it necessitates all parent tree-nodes to be predicted as non-leaf nodes. As depicted in Figure 11, certain strokes may be incorrectly connected or missing.

6 CONCLUSION

In this paper, we presented a novel high-efficiency font synthesis approach, QT-Font. To enhance representation efficiency, we proposed a sparse glyph representation based on quadtree and a filling algorithm that renders a point cloud into a glyph image. Compared to existing representations, our method offers uniqueness and lower representation complexity. Moreover, we designed QT-Font using dual quadtree graph networks to reduce computational complexity. Additionally, we employed a content-aware pooling module to constrain the additional cost caused by noise during the diffusion process. Extensive experiments demonstrated that QT-Font can rapidly generate high-resolution and superior-quality glyph images with fewer parameters and computation requirements.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (Grant No.: 62372015), Center For Chinese Font Design and Research, and Key Laboratory of Intelligent Press Media Technology.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*. PMLR, 40–49.

Haruka Aoki and Kiyoharu Aizawa. 2022. SVG Vector Font Generation for Chinese Characters with Transformer. *arXiv preprint arXiv:2206.10329* (2022).

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems* 34 (2021), 17981–17993.

Samaneh Azadi, Matthew Fisher, Vladimir G Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. 2018. Multi-content gan for few-shot font style transfer. In *CVPR*. 7564–7573.

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

Chia-Hao Chen, Ying-Tian Liu, Zhifei Zhang, Yuan-Chen Guo, and Song-Hai Zhang. 2023. Joint Implicit Neural Representation for High-fidelity and Compact Vector Fonts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5538–5548.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 45–54.

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

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. 2020. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 31–44.

Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems* 34 (2021), 8780–8794.

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

Chris Green. 2007. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*. 9–18.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. 2022. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10696–10706.

Haibin He, Xinyuan Chen, Chaoyue Wang, Juhua Liu, Bo Du, Dacheng Tao, and Yu Qiao. 2022. Diff-Font: Diffusion Model for Robust One-Shot Font Generation. *arXiv preprint arXiv:2212.05895* (2022).

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems* 33 (2020), 6840–6851.

Yaoxiong Huang, Mengchao He, Lianwen Jin, and Yongpan Wang. 2020. RD-GAN: few/zero-shot Chinese character style transfer via radical decomposition and rendering. In *European conference on computer vision*. Springer, 156–172.

Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2017. DCFont: an end-to-end deep Chinese font generation system. In *SIGGRAPH Asia*. ACM, 22.

Yuxin Kong, Canjie Luo, Weihong Ma, Qiyuan Zhu, Shenggao Zhu, Nicholas Yuan, and Lianwen Jin. 2022. Look Closer to Supervise Better: One-Shot Font Generation via Component-Based Discriminator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13482–13491.

Zhouhui Lian and Yichen Gao. 2022. CVFont: Synthesizing Chinese Vector Fonts via Deep Layout Inferring. In *Computer Graphics Forum*. Wiley Online Library.

Qisheng Liao, Zhinuo Wang, Muhammad Abdul-Mageed, and Gus Xia. 2023. CalliPaint: Chinese Calligraphy Inpainting with Diffusion Model. *arXiv preprint arXiv:2312.01536* (2023).

Yitian Liu and Zhouhui Lian. 2023. DeepCalliFont: Few-shot Chinese Calligraphy Font Synthesis by Integrating Dual-modality Generative Models. *arXiv preprint arXiv:2312.10314* (2023).

Ying-Tian Liu, Yuan-Chen Guo, Yi-Xiao Li, Chen Wang, and Song-Hai Zhang. 2022. Learning implicit glyph shape representation. *IEEE Transactions on Visualization and Computer Graphics* (2022).

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

Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengteng Huang, and Wenyu Liu. 2017. Auto-encoder guided GAN for Chinese calligraphy synthesis. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1. IEEE, 1095–1100.

Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. 2022. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751* (2022).

Wei Pan, Anna Zhu, Xinyu Zhou, Brian Kenji Iwana, and Shilin Li. 2023. Few shot font generation via transferring similarity guided global style and quantization local style. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 19506–19516.

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

Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. 2020. Few-shot Font Generation with Localized Style Representations and Factorization. *arXiv preprint arXiv:2009.11042* (2020).

Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. 2021. Multiple Heads are Better than One: Few-shot Font Generation with Multiple Localized Experts. *arXiv preprint arXiv:2104.00887* (2021).

Pradyumna Reddy, Zhifei Zhang, Zhaowen Wang, Matthew Fisher, Hailin Jin, and Niloy Mitra. 2021. A multi-implicit neural representation for fonts. *Advances in Neural Information Processing Systems* 34 (2021), 12637–12647.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10684–10695.

Danyang Sun, Tongzheng Ren, Chongxun Li, Hang Su, and Jun Zhu. 2017. Learning to write stylized chinese characters by reading a handful of examples. *arXiv preprint arXiv:1712.06424* (2017).

Licheng Tang, Yiyang Cai, Jiaming Liu, Zhibin Hong, Mingming Gong, Minhu Fan, Junyu Han, Jingtuo Liu, Errui Ding, and Jingdong Wang. 2022. Few-Shot Font Generation by Learning Fine-Grained Local Styles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7895–7904.

Yuchen Tian. 2017. zi2zi: Master chinese calligraphy with conditional adversarial networks, 2017. *Retrieved Jun* 3 (2017), 2017.

Chi Wang, Min Zhou, Tiezheng Ge, Yuning Jiang, Hujun Bao, and Weiwei Xu. 2023. CF-Font: Content Fusion for Few-Shot Font Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1858–1867.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions On Graphics (TOG)* 36, 4 (2017), 1–11.

Peng-Shuai Wang, Yang Liu, and Xin Tong. 2022. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

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

Jane Wilhelms and Allen Van Gelder. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)* 11, 3 (1992), 201–227.

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

Zeqing Xia, Bojun Xiong, and Zhouhui Lian. 2023. 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*. 1848–1857.

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

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 4541–4550.

Zhenhua Yang, Dezhi Peng, Yuxin Kong, Yuyi Zhang, Cong Yao, and Lianwen Jin. 2023. FontDiffuser: One-Shot Font Generation via Denoising Diffusion with Multi-Scale Content Aggregation and Style Contrastive Learning. *arXiv preprint arXiv:2312.12142* (2023).

Jinshan Zeng, Qi Chen, Yunxin Liu, Mingwen Wang, and Yuan Yao. 2021. Strokegan: Reducing mode collapse in chinese font generation via stroke encoding. In *proceedings of AAAI*, Vol. 3.

Yexun Zhang, Ya Zhang, and Wenbin Cai. 2018. Separating style and content for generalized style transfer. In *CVPR*. 8447–8455.

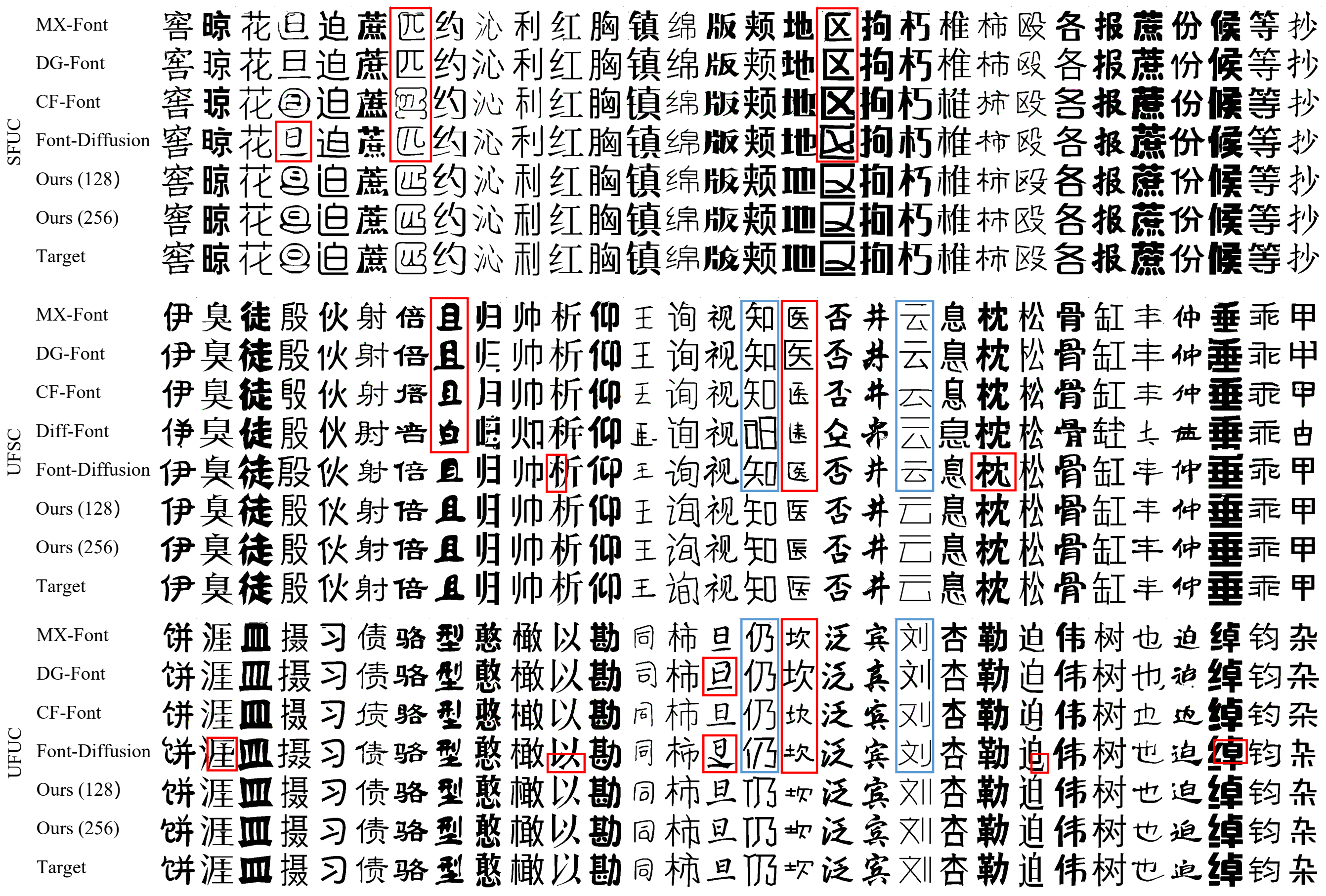

Figure 8: Comparison of synthesis results obtained by QT-Font and other SOTA methods. We highlight the failure cases.

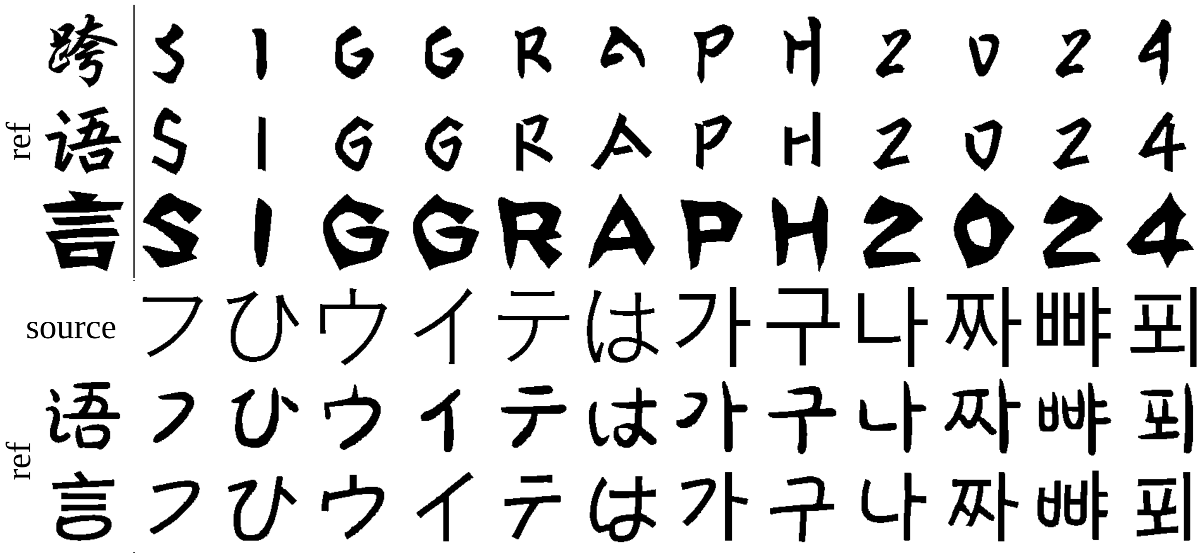

Figure 9: Cross-language synthesis results of our method.

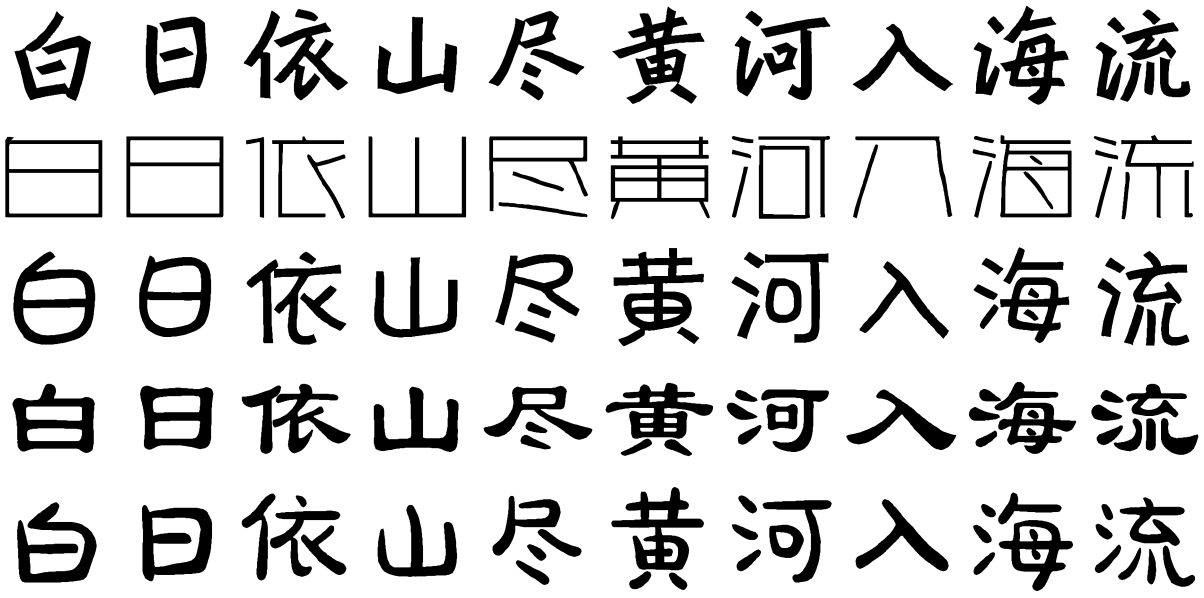

Figure 10: Synthesized results in resolution of 512×512.

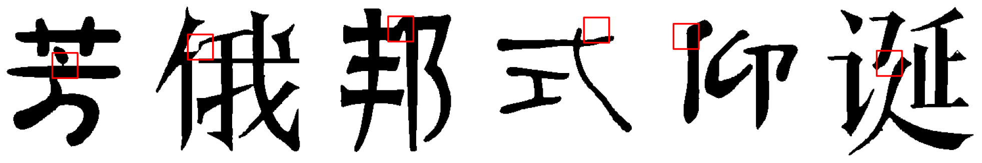

Figure 11: Some failure cases of the proposed method.

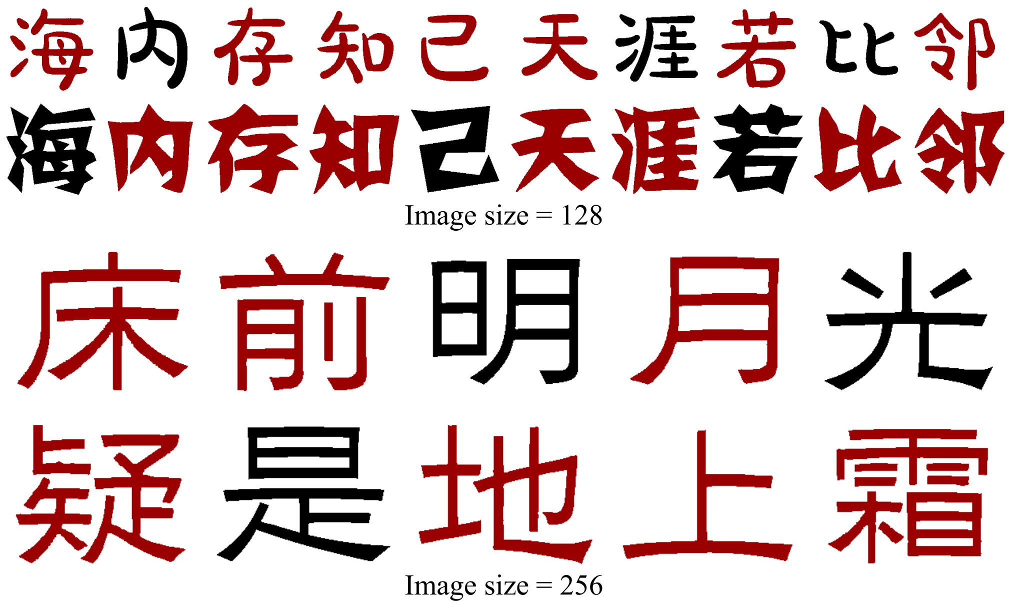

Figure 12: Texts render with the synthesized glyph images obtained by our method. Synthesized glyphs are highlighted in red. The proposed model can generate glyphs in a consistent style with the corresponding ground truth.