Neural Outline Cache for Real-time Anti-aliasing Font Rendering

Jiashuaizi Mo[13], **Sang-Woon Jeon**[4], **Hua Wang**[5], **Xiangqi Chen**[13], **Yanchao Wang**[13], **Minglu Li**[1], **Zhonglong Zheng**[123*]

[1]Zhejiang Key Laboratory of Intelligent Education Technology and Application, Zhejiang Normal University, Jinhua, Zhejiang 321004, China
[2]China-Mozambique "Belt and Road" Joint Laboratory on Smart Agriculture, Zhejiang Normal University, Jinhua, Zhejiang, 321004, China
[3]School of Computer Science and Technology, Zhejiang Normal University, Zhejiang, 321004, China
[4]Department of Electrical and Electronic Engineering, Hanyang University, Seoul, 04763, South Korea
[5]Institute for Sustainable Industries and Liveable Cities, College of Engineering and Science, Victoria University, Melbourne, VIC, 8001, Australia
molasses@zjnu.edu.cn, sangwoonjeon@hanyang.ac.kr, hua.wang@vu.edu.au,
{zjnu_cxq, yanchaowang, mlli, zhonglong}@zjnu.edu.cn

Abstract

Neural textures have emerged as pivotal assets in next-generation neural rendering pipelines. However, hardware limitations and programming interface constraints lead to suboptimal performance in multi-instance real-time rendering scenarios. This bottleneck becomes particularly acute for texture-intensive tasks such as font rendering. To address this, we propose Neural Outline Cache (NOC), a novel neural font texture supporting real-time anti-aliased rendering and procedural editing within modern neural graphics pipelines. NOC's lightweight network leverages multi-resolution hash encoding to cache spline-derived SDFs, delivering anti-aliased rendering via standard graphics pipelines. For massive-instance scalability, our cache buffer layout (CBL) and batch-fused inference (BFI), tailored for NOC, mitigate neural texture streaming bottlenecks. We constructed an evaluation dataset using five font styles. In offline rendering, our proposed method achieves overall average results of 57.35 dB PSNR, 0.998 SSIM, and 1.1584e-3 pixel RMSE, while maintaining approximately 0.5ms frame latency with 500 real-time instances. To demonstrate its versatility, we integrated a procedural editor for visual effects editing of NOC textures. These results all prove that NOC is a reliable, production-ready neural asset.

Introduction

Neural representation as rendering asset is a novel concept, referring to a class of deep learning-based methods designed for rendering pipeline data, including parametrization, compression and generalization. Unlike general neural representations, neural rendering assets prioritize integration and enhancement with standardized rendering workflows. This is where all the challenges begin, with higher and more standardized requirements for neural representation equivalence, versatility and editability.

Font is one of the key rendering assets that encounter these challenges. Compared to objects or scenes, deep learning-based font representation methods primarily focus on style transfer and interpolation prediction. These approaches achieve excellent performance metrics and visual appearance on benchmark datasets. However, due to their complex and redundant model designs, they are often not suitable to be applied in real-time systems. The integration with mainstream graphics pipelines or rendering engines remains insufficiently explored.

Challenges

Modern digital fonts are often designed using spline curves to outline scalable geometry. Fonts, being a typical vector graphic, demand higher rendering precision. Unlike general object and scene rendering, flaws in fonts are more easily observed.

Utilizing neural networks to represent fonts is essentially a form of parameterized encoding. Previous methods for extracting features from font images generalize font shape characteristics but suffer from excessive redundancy in model parameters. Methods that extract information from strokes and distance fields offer stronger geometric representation, but often rely on additional differentiable rendering layers to generate the image. Most previous works have focused primarily on evaluation on benchmark datasets. Although significant progress has been made in rendering quality, few studies consider how a neural font should function as a rendering asset for interactive real-time renderer. When these neural representation methods are integrated into rendering pipelines and creative workflows, their interaction often lacks intuitive editing capabilities.

Thus, we summarize several challenges faced by neural font representation, while also setting reference goals for the method we propose.

Lightweight model for high-precision geometry. The spline geometry of the font itself is the core of the neural representation, and geometric precision should be the top priority for implicit font representation, rather than interpolating for style generalization. Once an efficient representation is obtained, training additional modular sub-networks for styl-

*Corresponding Author.
Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ization and interpolation becomes more manageable.

Efficient multi-instance rendering. Text rendering is fundamentally a multi-instance task. Therefore, neural font representations should also consider handling multiple instances, but this aspect is often overlooked. In real-time applications, such as high-fidelity games, most computational resources are devoted to complex scene rendering, while text often serves as an attachment to the scene, occupying only a small portion of the available resources. Similarly, neural font rendering should minimize frame time usage, allowing sufficient time for other computations.

Procedural generation versus generative model. Style transfer and prompt generation are rapidly advancing, while procedural generation can more clearly and reliably express the visual artist's intuitive edits. How to effectively integrate procedural nodes with neural representations remains an area worth exploring. Thus, procedural methods for enabling interactive editing of neural fonts offer considerable practical value, while also presenting significant challenges.

Contributions

Different from previous work focused on prototyping, we consult with rendering engineers and professional artists for insights and perspectives to guide our development. In this paper, our work addresses two key challenges in applying neural font representations to real-time rendering. Building upon this, we demonstrate a novel interactive editing approach that integrates neural representations with procedural generation. The main contribution of this paper are concluded as following.

New implicit neural-represented texture for fonts. We propose a new implicit neural representation for caching font original spline curves, which can accurately learn geometric textures at fast caching speed. Our method demonstrates excellent performance metrics and visual quality in both offline and real-time rendering tasks.

Optimized for real-time rendering system. We design a fused pipeline that collaborates with the graphics pipeline, against the original serial inference approach. This pipeline performs dense inference of massive NOC models in a single pass, enhancing memory bandwidth during both the warm-up and inference stages. As a result, the real-time rendering phase achieves extremely fast speeds.

Industry standard visual effects creation supports. We develop and integrate a serial of industry-standard creation tools for NOC, including custom shading nodes, procedural content generation graph and generative model for filling materials. With these familiar interactive tools, technical artists and visual effects creators can quickly get started and use our method for production.

Related Works

Font Rendering in Computer Graphics

Spline-based font representation stands as an established industry standard, such as TrueType(Inc. 2019), OpenType(Typography 2024) and PostScript(PostScript 1999), with such font data enjoying widespread adoption across both creative and gaming domains.

Anti-aliased font rendering involves rasterizing vector spline data into screen-space images, ensuring complete interior fills and crisp outlines at varying scales during real-time viewport interactions. Designing anti-aliased font rendering algorithms requires balancing visual sharpness against smoothness while maintaining real-time performance and minimizing computational overhead.

Font meshes. Early font rendering methods used spline data to triangulate and generate meshes. The level of mesh subdivision directly determines the quality of font rendering. Due to the limitations of early rendering pipeline designs, dynamic font mesh subdivision was highly inefficient. Mesh shading pipeline (Kubisch 2018) is a more recent developed graphics pipeline that improves flexibility and efficiency of geometry processing. While a technical demonstration(Bastian Kuth 2024) implements this rendering pipeline, broader compatibility concerns have limited widespread adoption of the approach.

Font bitmaps and textures. A more efficient practice involves precomputing fonts as bitmap textures for rendering. This approach simplifies mesh-based fonts into a single rectangle (composed of two triangles), significantly reducing vertex count during rendering. The resulting visual quality is directly dependent on bitmap resolution. Furthermore, font textures leverage the texture processing unit's rapid interpolation sampling capabilities to provide limited anti-aliasing. Due to their simplicity, efficiency, and optimized hardware support, bitmap font textures have become the de facto standard asset format across diverse applications.

SDF-based fonts. The SDF-based font (Green 2007) is an efficient technique widely used in games and art design for improved rendering of glyphs with curved and linear elements. SDFs generate a distance field from a high-resolution font image, stored in a lower-resolution texture. During rendering, hardware alpha testing on this texture determines pixel visibility, with interpolation handled by the texture processing unit. While single-channel SDFs can cause corner distortion at sharp angles, multi-channel SDFs (Chlumsky 2015) refine these angles through texture composition. Due to high quality and efficiency, the SDF-based font becomes the best practice of anti-aliasing font rendering and gets well supports in mainstream game engine(Unity 2024; EpicGames 2024).

Implicit Neural Representations

Radiance caching. The neural radiance field(Mildenhall et al. 2021) has established the era of the neural representation, being proposed in novel-view synthetics task. It trains a deep MLP network with residual connection to map the spatial positions in single scenario to the radiance along the emitted view lights. Instant neural graphics primitives (NGP) (Müller et al. 2022) encodes 3D positional encodes gets evolved by formulating as a trainable hash table in a multi-resolution sampling manner. Its MLP model (Müller 2021) is re-implemented to be a lightweight and fully-fused form to map the grid encoding to on-grid features that can be fast interpolated. As a result, the training and inference speed of NGP are greatly improved and seamlessly combined. This implementation is usually embedded in a real-

time rendering system as an extension. 3D Gaussian Splatting (3DGS) (Kerbl et al. 2023) approximates radiance fields via mixtures of 3D Gaussian primitives, rasterized through alpha-blended splatting.

Implicit surfaces. To modeling the object surfaces, NeuS(Wang et al. 2021) uses fully-connected network to map the spatial positions to the surface distance values and the surface colors, establishing using neural SDF to represent surface and boundary. NeuS2(Wang et al. 2023a) improves training speed via hashing encoding and secondary gradient solver. Other typical works, like Neuralangelo(Li et al. 2023) and NGLOD(Takikawa et al. 2021), focus on subsequent improvements on aspect of accuracy and level of details respectively.

Random-access neural textures. Texture in GPU memory is a block specified memory buffer that supports highly hardware-accelerated sampling interpolation, random access and mipmap levels. Using the similar architecture and grid encoding as NGP does, neural texture compression (NTC) (Vaidyanathan et al. 2023) was proposed to compress high-resolution texture resources with multiple levels of details. Its feature pyramid, implemented using multi-resolution hash encoding, enables random access and feature interpolation in neural texture representations.

Neural font representations. There are some typical related works that learn font feature via CNN and VecFontSDF(Xia, Xiong, and Lian 2023) constructs a deep convolutional network as encoder-decoder structure to learn font SDFs. Multi-implicit neural representation (Reddy et al. 2021) uses deep CNN-based generative adversarial network (Goodfellow et al. 2020) to encode SDF. JointFontGAN(Xi et al. 2020) also adopts U-Net based GAN network to get geometry-aware feature extraction for fonts. JointMultiReps(Chen et al. 2023) builds a dual sub-network framework to learn SDF and guide corner rendering. The above works learning SDFs have proved high quality across varying target rendering. For the network framework, DeepVecFont(Wang et al. 2023b) and FontTransformer(Liu and Lian 2023) have done well exploration works using Vision Transformer-based framework.

Within neural network-based font representations, typeface feature extraction and style transfer dominate current research. (Balashova et al. 2019; Berio et al. 2022; Campbell and Kautz 2014; Atarsaikhan et al. 2017; Azadi et al. 2018; Liu and Lian 2023; Wang et al. 2023b). However, scant attention has been paid to neural fonts as reusable assets or neural texture workflows. Consequently, real-time multi-instance rendering of neural fonts remains an open challenge.

Methodology

Outline Contour Representation

Signed distance function. Similar to the implicit surface of a 3D object, the 2D outline contour curve $\mathcal{E}_0$ of a glyph is represented by the level set of its SDF, that is

$$\mathcal{E}_0 = \{\mathbf{x} \in R^2 | f(\mathbf{x}) = t_o\}. \tag{1}$$

Any arbitrary point on this level set lies on the spline curve. t_o is the threshold value from 0 to 1 which defines actual outline to be rendered, which is used for alpha testing in Procedure 6 to achieve anti-aliasing effects along the edge.

$$\mathbf{p}(t) = (1-t)^2 \cdot \mathbf{p}_0 + 2t(1-t) \cdot \mathbf{p}_1 + t^2 \cdot \mathbf{p}_2 \tag{2}$$

TrueType's (Inc. 2019; Typography 2024) curves are represented as quadratic B-splines. Each spline can be equivalently expressed as a series of quadratic Bézier curves. As Eq.2 formulates, each of these is defined by three outline control points $\mathbf{p}_0, \mathbf{p}_1$ and $\mathbf{p}_2$. Varying the parameter t from 0 to 1 produces all the points $\mathbf{p}$ on the curve defined by the control points. In this way, it eliminates brute-force spatial sampling and fitting. Instead, directly sample the contour curves and their neighborhoods to cache the geometric features.

Sampling guidance and acceleration. The area where the SDF value is greater than t_o is defined as the interior region $\mathcal{E}_-$; And define the exterior area $\mathcal{E}_+$, where the SDF value is less than t_o. The three sets $\mathcal{E}_-, \mathcal{E}_0, \mathcal{E}_+$ described above together form a valid region of one glyph. Denote this region as $\mathcal{E}_* = \mathcal{E}_- \cup \mathcal{E}_0 \cup \mathcal{E}_+$ and extract this region to use as an auxiliary binary mask $\mathbf{M} = binarize(\mathcal{E}_*)$. During training, this mask can be used to filter sampling points, ensuring that only those within the valid region are selected, thereby reducing unnecessary sampling. During inference, the mask helps discard pixels outside the glyph's valid area, ensuring a clean background and a seamless filling of the rendering area.

Neural Outline Cache

Neural Outline Cache (NOC) is designed to be an implicit neural geometry proxy learns the geometric distance metrics around the edges of spline curves. Here, "cache" indicates that this method will reside in memory to accelerate read and write access, providing a data view.

Model design. For each sample point $\mathbf{x} \in \mathcal{E}_0$ along the outline curves in 2D space, we encode the spatial information of sample points $\mathbf{p}$ inside its neighborhood $U(\mathbf{x}, \rho)$ with a learnable multi-resolution hash encoding $h_\Omega(\mathbf{p})$. Then, a fully-connected network maps the encoded position to its SDF value d, which caches the distance mapping between the contour curve and neighborhood pixels. More detailed, the model is formulated as,

$$\mathbf{e} = h_\Omega(\mathbf{p}), d = f_\Theta(\mathbf{e}), \tag{3}$$

where $\mathbf{p} \in U(\mathbf{x}, \rho)$, here $U(\mathbf{x}, \rho)$ denotes the $\mathbf{x}$-centered with a radius of ρ neighborhood point set in a specific 2D grid. The $f_\Theta(\cdot)$ denotes the two-layer lightweight MLP with its trainable parameters Θ. Unlike other methods used in surface reconstruction tasks, mapping the color component is not required for the model unless a multiple-channel output (Chlumsky 2015; Reddy et al. 2021) is necessary, although it remains optional. We can still apply shading to fonts and create visual effects within the hardware graphics pipeline.

Supervision. To train NOC model, we optimize the model to approximate the ground truth geometry proxy. For each points batch, the target is to minimize the average reconstruction error between the distance values and the cache

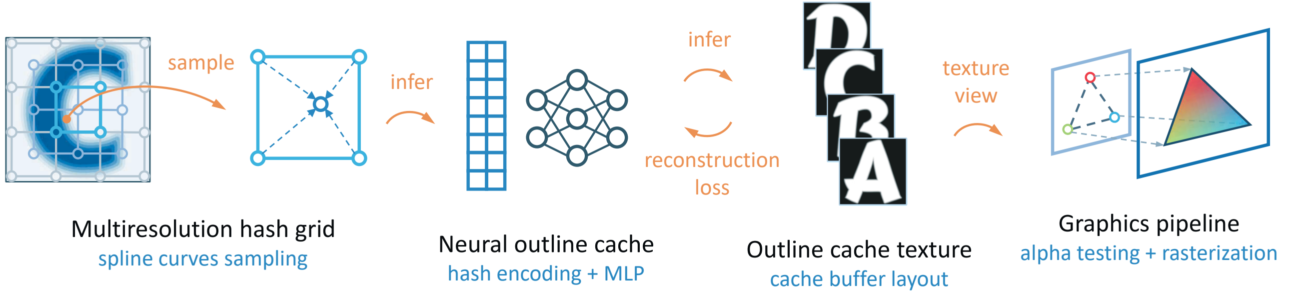

Figure 1: The overview of NOC pipeline. The distance values are sampled from the original B-spline curves within multiresolution grids. A two-layer MLP network maps the encoded coordinates to the distance values, each layer of this model has 64 neurons with ReLU activation function. L1 reconstruction loss is computed as the training supervision. We design a novel buffer layout (CBL) and batch-fused inference (BFI) method to make NOC texture be rapidly available as render resources. Once the texture addressing and rasterization pipeline gets ready, the fonts can be anti-aliasing rendered to screen via hardware alpha-testing and texture interpolation.

values at the points. When conducting offline rendering or being edited, the loss between the image rendered by current cache model and the reference image can provide effective pixel-wise supervision. Consider the above two factors, the full loss function is collected as:

$$L_{rec} = \frac{1}{m} \sum_{i} loss(\hat{\mathbf{p}}_i, \mathbf{p}_i) \tag{4}$$

where m denotes the batch size of sample points $\mathbf{p}_i$. Let $\hat{\mathbf{I}} \in R_{H\times W}$ and $\mathbf{I} \in R_{H\times W}$ be the rendered image and the ground truth image respectively, $L_{image} = \frac{1}{WH}\sum_i^H \sum_j^W loss(\hat{\mathbf{I}}_{i,j}, \mathbf{I}_{i,j})$; H is height of the output image, W is the width of the output image. The function $loss(\cdot)$ are computed using smooth L1 loss function.

Rasterization within Graphics Pipeline

Given a trained NOC model or an in-memory one, the goal is to infer the NOC texture and to render the glyph onto image plane or screen space.

From inference to rasterization. First, input the normalized integer coordinates $\mathbf{g}$ in 2D regular grid $\mathbf{G} \in Z^2$ into the cache model $f_\Theta(\mathbf{g})$ to obtain the cached distance values to be texture $\mathbf{D}$. To enable sparse sampling, the auxiliary mask $\mathbf{M}$ can help to filter out sample grid coordinates not in the contour range. Then, the inferred results are marked as texture view and visible to the hardware rasterization pipeline. This procedure is denoted as $render(\cdot)$, which means to render the pixels on the imaging plane. Thus, the font rasterization process can be formulated as follows:

$$\mathbf{D} = \{\mathbf{d} = f_\Theta(\mathbf{g}) \mid \mathbf{g} \in \mathbf{G} \cap \mathbf{M}\}, \mathbf{I} = render(\mathbf{D}, \mathbf{M}). \tag{5}$$

The sizes of the outline cache texture $\mathbf{D}$, coordinates gird $\mathbf{G}$ and auxiliary mask $\mathbf{M}$ are identical. The output image $\mathbf{I}$ can be rendered at resolutions up to $R\times$ that of the outline cache texture, where R is the pixel interpolation ratio as the hyper-parameter of assets. And the actual rendered size of the texture onto screen space adjusts according to the camera viewport during real-time rendering.

Alpha testing and pixel interpolation. The main process of $render(\cdot)$ is to apply alpha testing and pixel interpolation on the available texture resource (Green 2007; Chlumsky 2015). For each pixel $\mathbf{I}_{(i,j)}$, we compute its screen pixel range to obtain its color blending opacity by Procedure 6.

$$\begin{aligned} d_{(i,j)} &= sample2D(\mathbf{D}, u, v) \\ o_{(i,j)} &= \max\{t_o \cdot L_{tex;(u,v)}, 1.0\} \cdot normalize(d_{(i,j)}) \\ \mathbf{I}_{(i,j)} &= blend(\mathbf{c}_{fore}, \mathbf{c}_{back}, o_{(i,j)}) \end{aligned} \tag{6}$$

The operator $sample2D(\cdot, u, v)$ is the hardware texture linear sampler with the normalized sampling coordinates (u, v). The alpha threshold t_o is commonly set to 0.5 or adjusted accordingly for edit. L_{tex} denotes the actual texel size on screen space. To obtain the pixel color, blend the foreground color $\mathbf{c}_{fore}$ (font color) and background color $\mathbf{c}_{back}$ with this opacity value. The blending behavior can change by changing the related settings of the program instead of the linear interpolation.

Cache Buffer Layout and Batch-Fused Inference

Limited by hardware architecture and graphics programming frameworks, utilizing neural rendering assets in modern rendering pipelines still presents several challenges. One of the primary difficulties is the computational scheduling of hybrid neural and graphics rendering operations. Another challenge is designing efficient data delivery mechanisms to supply usable data to the rendering pipeline. The neural representation model typically represents a single object or scene, where the entire scene can be inferred using a single model. However, when rendering text, the rendered objects comprise multiple character instances. For neural font representation, this necessitates sequential inference across multiple models to render a single frame.

To overcome these, we tailor a new memory layout and inference method for NOC, illustrated in Figure 2, aimed at

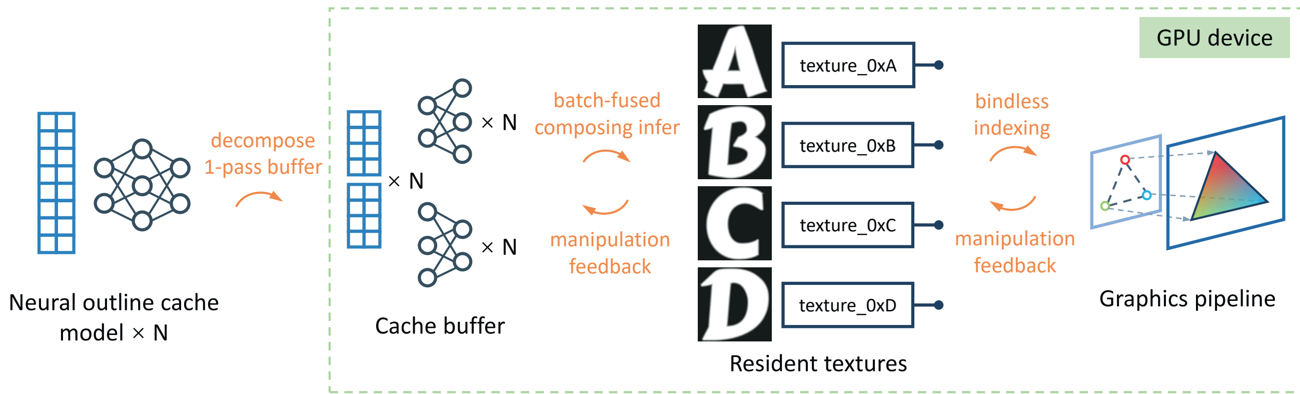

Figure 2: Model decomposition, cache buffer layout and batch-fused inference for real-time optimization. The figure illustrates the rasterization workflow from the model inference to the screen image. The CBL organizes model parameters at the same layer into batches, then executes fused kernels on these batches. The inference results are saved into a pre-allocated resident texture memory pool. The graphics pipeline can perform bindless addressing to access the textures for rendering.

the rapid submission of rendering resources to the graphics pipeline and receiving feedback from edit operations.

Model decomposition and cache buffer layout. Single NOC instance has a small parameter quantity, and performing inference sequentially of this model is not able to fully utilize the memory bandwidth. The frequent data transfer to GPU global memory further degrades the inference process of multiple models. Inspired by the Atlas texture, the cache buffer layout is designed for NOC to improve memory bandwidth utilization to optimize inference time. We denote the trainable parameters of the hash positional encoder $h(\cdot)$ and the lightweight network $f(\cdot)$ as Θ and Ω. The decomposition of model cluster can be formulated as

$$\begin{aligned} \Theta_i &\to \{\theta_{i,1}, \ldots, \theta_{i,j}, \ldots, \theta_{i,N_{\text{grid}}}\} \\ \Omega_i &\to \{\omega_{i,1}, \ldots, \omega_{i,k}, \ldots, \omega_{i,N_{\text{layer}}}\} \end{aligned} \tag{7}$$

Specifically, decompose a group of NOC models into several components, including hash encoding parameters and two layers of MLP weight matrices. Then reorganize the models parameters in batches in the same layer and mark them with the same memory alignment to optimize indexing and cache access. If memory allocation allows, this cache buffer can be sent to the GPU global memory through one-pass buffering.

Batch-fused inference. During the actual inference, we fuse the weights computation in the cache buffer in layer order with parallel threads by slicing them to fit the shared memory size. The reason for this approach is that the workflow reduces the frequency of synchronization or barriers, so that the work groups make best effort to deliver the computation results to the resident memory. Threads dispatching and shared memory access are carefully adjusted to better accelerate the GEMM computation in our custom batch-fused kernel. We denote the fuse kernel as $\text{fuse}(\cdot)$, such that the batch-fused inference procedure can be roughly formulated as

$$\begin{aligned} \{h_{\Theta_1}, \ldots, h_{\Theta_{N_{\text{cache}}}}\} &= \text{fuse}(\{h_{\theta_{1,1}}, \ldots, h_{\theta_{N_{\text{cache}}, N_{\text{grid}}}}\}), \\ \{f_{\Omega_1}, \ldots, f_{\Omega_{N_{\text{cache}}}}\} &= \text{fuse}(\{f_{\omega_{1,1}}, \ldots, f_{\omega_{N_{\text{cache}}, N_{\text{layer}}}}\}). \end{aligned} \tag{8}$$

The computation results are transformed to the specified texture format in a pre-allocated memory block. This memory block uses uniform bindless resource addressing across the compute and graphics pipelines with a low access cost. Once the inference of a NOC model group is completed, the textures on this memory block are instantaneously available as the render resource then to be used in graphics pipeline by switching to texture view.

Manipulation feedback. Through residency and view switching, the forward chain is constructed from the cache buffer to the graphics pipeline. To enable interactive editing, we establish an additional connection between the rendering frame buffer and the cache texture. Considering compatibility and real-time performance, we adopt the deferred rendering approach to compose. Specifically, we overlay the font-rendering frame buffer with incremental textures to generate diverse visual effects. Manipulation feedback from the graphics pipeline is now able to directly backpropagate to the texture memory pool and the cache buffer through the frame. These feedback data efficiently provides geometric and visual modifications, all of which can produce valid losses or deltas to update the corresponding texture in pool or to fine-tune the model weights in the cache buffer.

Experiments

Dataset introduction. Five representative fonts with diverse design styles are selected to generate training data in order to comprehensively evaluate the performance of our method across varying levels of complexity. *Arial*(Nicholas and Saunders 2017) is a widely used sans-serif typefaces known for its clean, modern appearance. *Centaur*(Bruce 1992) is a renowned serif typefaces frequently used in newspaper and book printing for its readability and classic

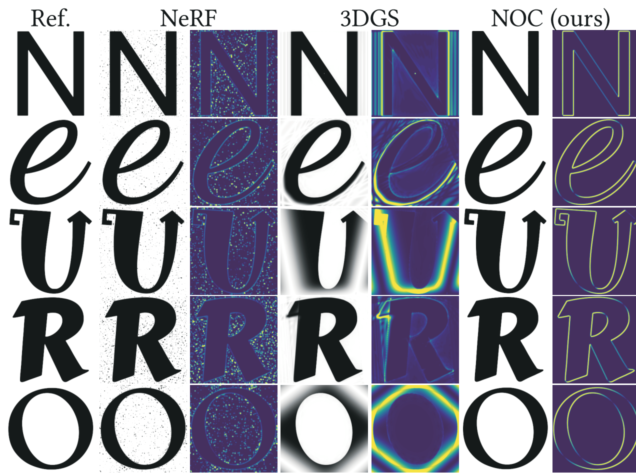

Figure 3: Offline rendering results and residual maps. We assembled the word *Neuro* using rendered results and residual maps, each selected from a distinct data subset. NOC consistently maintains clear and well-defined contours, as well as zero-error internal and external fills, across a diverse range of font styles. Further rendering results are available in the supplementary materials (see Appendix).

style. *CarterOne*(Adams 2011) is selected for testing bold and heavy designs due to its large pixel coverage. *DancingScript*(Impallari 2019) is a fluid and elegant handwritten typeface for evaluating curve rendering quality. *Halloween*(Aradea 2021) is a decorative font with distinctive, artistic style. All the training contour data is obtained B-spline from TrueType format.

Implementation details. The core model was prototyped using the PyTorch framework for both training and inference. To maximize hardware compatibility, the rendering engine and application feature a custom compute shader implementation of our CBL and BFI optimizations, while maintaining a conventional vertex shader and pixel shader rendering pipeline. All benchmarks were performed on a workstation with a 14-core Intel Core i5-13600K CPU and an NVIDIA GeForce RTX 3090 GPU.

Offline Rendering

In offline rendering evaluation, NOCs are trained with the dataset for 3,000 iterations. The Adam optimizer learning rate is initially set to 1e-2, with a decay factor of 0.1 every 1,000 iterations. We set the base sampling resolution to 16, the resolution multiplier to 1.5 per level, and the hash encoding level to 4. For rendering, the size of the texture is set to 64×64, the maximam pixel interpolation ratio R is set to 8, and the render target resolution is set to 512×512 per instance.

Learning accuracy analysis. We first evaluate pixel RMSE, PSNR and SSIM metrics for the learned geometry proxies which indicates the high learning accuracy of our settings. The average metrics of the results are collected in Table 1. Pixel error metrics are scaled by 10^3 to improve the visibility of significant digits. Low pixel error and high PSNR indicate that the our method effectively captures contour features and maintains a high level of consistency between the modeled geometric proxy and the original contour shape. Additionally, the high SSIM score underscores that the model is particularly strong in preserving the vector geometric structure.

Rendered results analysis. We show a group of representive rendered results for comparison to qualitatively evaluate our method, as shown in Figure 3. The absolute error maps between the rendered results and the reference images are provided to highlight areas of pixel error. The results rendered by the our method maintain a high level of consistency with those of the reference method. Our method excels across all font styles, surpassing alternative neural textures in both rendering quality and robustness.

Due to its sampling sensitivity, the NeRF model (Mildenhall et al. 2021) exhibits significant noise, with some instances encroaching on contour pixels and resulting in jagged artifacts. Both the NeRF model and our method maintain uniform internal fills; however, our results consistently match the target color. Although 3DGS(Kerbl et al. 2023) demonstrates stronger representational capacity than NeRF, it fails to capture sharp edges under constrained primitive budgets. As evidenced by serif (letter O) and artistic (letter U) fonts, 3DGS texture inadequately reconstructs high-frequency edge signals and introduces chromatic noise across backgrounds. In contrast, our method maintains pixel-accurate consistency with reference vector glyphs across all font styles, delivering clean backgrounds and crisply defined edges.

Real-Time Anti-Aliasing Rendering

We designed a scene with a large number of character instances for evaluation in our real-time renderer, and set three different instance levels. The renderer's main viewport resolution is set to 1080p. Full inference and rendering are performed using the NOC model trained on all datasets. For the graphics pipeline state, frustum culling is disabled, the depth test is enabled, and no extra multi-sampling is applied.

Frame latency analysis. The all experimental metrics results are summarized in Table 2. As rendering instances scale, our method maintains ultra-high frame rates and sub-millisecond latency. In contrast, NeRF-based approach become impractical for real-time interaction due to prohibitive inference costs at large instance counts. While 3DGS sustains usable frame rates, its resource consumption leaves insufficient frame-time budget for auxiliary rendering tasks – a critical constraint in production font rendering pipelines.

Procedural Visual Effects Application

The visual effects of artistic typography are primarily created by manipulating the interior and exterior components defined by the geometric outline. Contour lines, internal fills, and external outlines can all be edited using integrated procedural tools and are synthesized in real-time by the renderer. The manipulation feedback mechanism is designed to bridge the gap between neural texture and intuitive interactive editing. To validate this mechanism and demonstrate

	NeRF MLP			3DGS			**NOC (ours)**		
	PSNR ↑	SSIM ↑	PxRMSE ↓	PSNR ↑	SSIM ↑	PxRMSE ↓	PSNR ↑	SSIM ↑	PxRMSE ↓
Arial	50.00	0.9989	1.7755	29.33	0.9134	34.0272	**57.63**	**0.9992**	**1.1139**
Carter1	47.10	**0.9984**	2.0940	29.63	0.9223	38.9008	**58.41**	0.9965	**1.1202**
Centaur	47.63	0.9990	1.9212	28.81	0.9159	27.9203	**57.18**	**0.9995**	**1.0898**
D.Script	46.00	0.9989	2.1275	32.59	0.9480	18.3129	**56.20**	**0.9994**	**1.2192**
Halloween	46.36	**0.9986**	2.2207	29.95	0.9250	22.3438	**57.15**	0.9959	**1.2493**
Overall	47.42	**0.9988**	2.0278	30.06	0.9249	28.3010	**57.35**	0.9980	**1.1584**

Table 1: Offline Rendering Quality Evaluation on Full Dataset

Method	Instances ↑	FPS↑	AA ratio ↑
NeRF MLP	500	4.6	✘
3DGS		331.5	✘
NOC (ours)		**1969.2**	8×
NeRF MLP	5,000	✘	✘
3DGS		253.9	✘
NOC (ours)		**1871.2**	8×
NeRF MLP	50,000	✘	✘
3DGS		55.2	✘
NOC (ours)		**258.6**	8×

Table 2: Real-time Rendering Performance Comparison

Figure 4: Art works with NOCs by procedural generation. Using NOC textures, the interior, outline, and exterior are edited with standard procedural PBR materials (Burley and Studios 2012; MaterialX 2024) and custom shader graphs to create artistic typography.

the flexibility of NOCs, we use standard procedural textures (MaterialX 2024) to fill the fonts, a custom shader pass for edge coloring, and a procedural shader graph to control the entire rendering process.

As shown in Figure 4, three visual effects samples are presented. The proposed manipulation feedback effectively help synthesize high-quality edits to the NOC instances. The shading node is the basis for obtaining colors for glowing and filling, or from procedural and generative textures. The material node can use both baked material images and procedural textures to fill the interior. The system supports managing extra lighting through shader graphs to achieve a wider range of material effects, such as highlights and sub-surface scattering.

Conclusion and Future Works

Neural-represented assets have demonstrated promising results on benchmark datasets, raising high expectations within the industry. However, during our investigation, we identified several concerns and criticisms regarding the integration of neural shading into standard industrial workflows. In response, we present Neural Outline Cache (NOC), a neural font texture learned from original vector splines with integrated procedural editing. We further propose a universal optimization for the real-time application of massive neural texture instances within this framework. Finally, we want to discuss the limitations and future work of this research.

Interpolation. Finding the optimal method for NOC interpolation remains an open question. We attempted batched NOC interpolation using a contrastive learning approach, while maintaining high geometric accuracy. However, due to the varying requirements of different font representations, standard pixel-level contrastive loss functions cannot cover the interpolation between all stylistic fonts. Further research is needed to explore geometry-aware contrastive loss functions or investigate alternatives to contrastive learning.

Generalization and generation. The NOC model establishes strong pixel-level generalization geometrically, but we do not discuss encoding and generalization between different stylistic NOCs in this paper. Given the rapid development of generative models, we believe further research into NOC-based font generation models is both meaningful and necessary. Rather than saving the generated results as a NOC model, we are more interested in conditionally generating NOC-based latents or model weights, which could completely eliminate the need to train a NOC from scratch and further improve efficiency.

Acknowledgments

This work was supported in part by the Project of China-Mozambique “Belt and Road” Joint Laboratory on Smart Agriculture under Grant 2024YFE0214000, in part by the National Natural Science Foundation of China under Grant 62272419 and 62320106006.

References

Adams, V. 2011. Carter One. https://fonts.google.com/specimen/Carter+One. [Accessed 10-11-2025].

Aradea, T. 2021. Halloween island font. https://www.creativefabrica.com/product/halloween-island/. [Accessed 10-11-2025].

Atarsaikhan, G.; Iwana, B. K.; Narusawa, A.; Yanai, K.; and Uchida, S. 2017. Neural font style transfer. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 5, 51–56. IEEE.

Azadi, S.; Fisher, M.; Kim, V. G.; Wang, Z.; Shechtman, E.; and Darrell, T. 2018. Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7564–7573.

Balashova, E.; Bermano, A. H.; Kim, V. G.; DiVerdi, S.; Hertzmann, A.; and Funkhouser, T. 2019. Learning A Stroke-Based Representation for Fonts. In *Computer Graphics Forum*, volume 38, 429–442. Wiley Online Library.

Bastian Kuth, M. O. Q. M. 2024. Font and vector art rendering with mesh shaders. https://gpuopen.com/learn/mesh_shaders/mesh_shaders-font_and_vector_art_rendering_with_mesh_shaders/.

Berio, D.; Leymarie, F. F.; Asente, P.; and Echevarria, J. 2022. Strokestyles: Stroke-based segmentation and stylization of fonts. *ACM Transactions on Graphics (TOG)*, 41(3): 1–21.

Bruce, R. 1992. Centaur font family. https://learn.microsoft.com/zh-tw/typography/font-list/centaur. [Accessed 10-11-2025].

Burley, B.; and Studios, W. D. A. 2012. Physically-based shading at disney. In *Acm Siggraph*, volume 2012, 1–7. vol. 2012.

Campbell, N. D.; and Kautz, J. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics (ToG)*, 33(4): 1–11.

Chen, C.-H.; Liu, Y.-T.; Zhang, Z.; Guo, Y.-C.; and Zhang, S.-H. 2023. Joint Implicit Neural Representation for High-fidelity and Compact Vector Fonts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5538–5548.

Chlumsky, V. 2015. *Shape decomposition for multi-channel distance fields*. Ph.D. thesis, Master's thesis, Czech Technical University. URL: https://dspace. cvut. cz . . . .

EpicGames. 2024. Font Materials and Outlines. https://dev.epicgames.com/documentation/en-us/unreal-engine/font-materials-and-outlines-in-unreal-engine.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

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

Impallari. 2019. Dancing Script. https://fonts.google.com/specimen/Dancing+Script. [Accessed 10-11-2025].

Inc., A. 2019. TrueType Reference Manual. https://developer.apple.com/fonts/TrueType-Reference-Manual/.

Kerbl, B.; Kopanas, G.; Leimkühler, T.; and Drettakis, G. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4).

Kubisch, C. 2018. Introduction to Turing Mesh Shaders. https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/.

Li, Z.; Müller, T.; Evans, A.; Taylor, R. H.; Unberath, M.; Liu, M.-Y.; and Lin, C.-H. 2023. Neuralangelo: High-fidelity neural surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8456–8465.

Liu, Y.; and Lian, Z. 2023. FontTransformer: Few-shot high-resolution Chinese glyph image synthesis via stacked transformers. *Pattern Recognition*, 141: 109593.

MaterialX, A. 2024. MaterialX - Home - materialx.org. https://materialx.org/index.html.

Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1): 99–106.

Müller, T.; Evans, A.; Schied, C.; and Keller, A. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4): 1–15.

Müller, T. 2021. NVlabs/tiny-cuda-nn: Lightning fast C++/CUDA neural network framework. https://github.com/NVlabs/tiny-cuda-nn.

Nicholas, R.; and Saunders, P. 2017. Arial font family. https://learn.microsoft.com/zh-tw/typography/font-list/arial. [Accessed 10-11-2025].

PostScript, R. 1999. Language Reference Third Edition, Adobe Systems Incorporated.

Reddy, P.; Zhang, Z.; Wang, Z.; Fisher, M.; Jin, H.; and Mitra, N. 2021. A multi-implicit neural representation for fonts. *Advances in Neural Information Processing Systems*, 34: 12637–12647.

Takikawa, T.; Litalien, J.; Yin, K.; Kreis, K.; Loop, C.; Nowrouzezahrai, D.; Jacobson, A.; McGuire, M.; and Fidler, S. 2021. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11358–11367.

Typography, M. 2024. OpenType specification (OpenType 1.9.1). https://learn.microsoft.com/en-us/typography/opentype/spec/.

Unity. 2024. Distance Field Overlay Shaders. https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/TextMeshPro/ShadersDistanceField.html.

Vaidyanathan, K.; Salvi, M.; Wronski, B.; Akenine-Möller, T.; Ebelin, P.; and Lefohn, A. 2023. Random-access neural compression of material textures. *arXiv preprint arXiv:2305.17105*.

Wang, P.; Liu, L.; Liu, Y.; Theobalt, C.; Komura, T.; and Wang, W. 2021. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*.

Wang, Y.; Han, Q.; Habermann, M.; Daniilidis, K.; Theobalt, C.; and Liu, L. 2023a. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3295–3306.

Wang, Y.; Wang, Y.; Yu, L.; Zhu, Y.; and Lian, Z. 2023b. Deepvecfont-v2: Exploiting transformers to synthesize vector fonts with higher quality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 18320–18328.

Xi, Y.; Yan, G.; Hua, J.; and Zhong, Z. 2020. Jointfont-gan: Joint geometry-content gan for font generation via few-shot learning. In *Proceedings of the 28th ACM International Conference on Multimedia*, 4309–4317.

Xia, Z.; Xiong, B.; and Lian, Z. 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.