Neural 3D Strokes: Creating Stylized 3D Scenes with Vectorized 3D Strokes

Hao-Bin Duan[1]   Miao Wang[1,2*]   Yan-Xun Li[1]   Yong-Liang Yang[3]

[1]State Key Laboratory of Virtual Reality Technology and Systems, SCSE, Beihang University

[2]Zhongguanchun Laboratory   [3]Department of Computer Science, University of Bath

Figure 1. We propose a method to stylize a 3D scene from multi-view 2D images using vectorized 3D strokes based on geometric primitives and splines. The four scenes from left to right are drawn with axis-aligned box, oriented box, ellipsoid, and cubic Bézier curve, respectively.

Abstract

We present Neural 3D Strokes, a novel technique to generate stylized images of a 3D scene at arbitrary novel views from multi-view 2D images. Different from existing methods which apply stylization to trained neural radiance fields at the voxel level, our approach draws inspiration from image-to-painting methods, simulating the progressive painting process of human artwork with vector strokes. We develop a palette of stylized 3D strokes from basic primitives and splines, and consider the 3D scene stylization task as a multi-view reconstruction process based on these 3D stroke primitives. Instead of directly searching for the parameters of these 3D strokes, which would be too costly, we introduce a differentiable renderer that allows optimizing stroke parameters using gradient descent, and propose a training scheme to alleviate the vanishing gradient issue. The extensive evaluation demonstrates that our approach effectively synthesizes 3D scenes with significant geometric and aesthetic stylization while maintaining a consistent appearance across different views. Our method can be further integrated with style loss and image-text contrastive models to extend its applications, including color transfer and text-driven 3D scene drawing. Results and code are available at http://buaavrcg.github.io/Neural3DStrokes.

---

* Corresponding author.

1. Introduction

Artistic style image creation, historically a domain requiring significant skill and time, has been revolutionized by neural network-based Style Transfer techniques [7, 8]. These methods usually separate, manipulate, and merge the content and style of images to create an artistic effect. Extending this to 3D scenes, however, is challenging due to complex geometries and appearance traits [28], and traditional convolutional neural network (CNN) methods [25] are not readily adaptable to 3D spaces. Recently, 3D implicit scene representations, particularly Neural Radiance Fields (NeRF), have become popular, since they are fully differentiable and easy to optimize. Efforts to apply artistic styles to NeRF [11, 15, 22] often involve separate NeRF training and style transfer, followed by integration style transfer loss from multiple views [3, 5, 32]. The fundamental approach continues to be the independent manipulation of content and style. Nevertheless, existing NeRF stylization techniques only alter the color aspects, leaving the density-and consequently the geometry—unchanged, leading to stylizations that lack geometric changes.

Style transfer methods in 2D and 3D primarily manipulate pixels or voxels to achieve artistic effects, differing from traditional art where artists use brushstrokes and a degree of randomness in brushes, materials, and colors. This traditional approach is time-consuming and skill-intensive. To automate it, research has explored image generation with

vectorized strokes [10], defined by position, color, size, and direction, and optimized to match target images. 3D painting tools like Google's Tilt Brush [4] use VR/AR to create art scenes with various brushes in virtual spaces. However, the field of automated stroke-based 3D art generation remains relatively unexplored.

In this work, we present a novel technique for transforming 2D images with known poses into stylized 3D scenes using vectorized 3D strokes. Our approach recreates 3D scenes that exhibit distinct geometric and appearance styles, emulating the stroke-by-stroke painting process employed by human artists. The vector-based nature of our system enables rendering at any desired resolution. It accommodates a range of 3D stroke styles and supports more intricate stylizations and generations through the use of style and semantic losses. The core of our method is a differentiable, vectorized 3D scene representation, which diverges from NeRF by using parameterized 3D strokes to synthesize 2D images at different viewpoints. This differentiable representation allows for the direct optimization of stroke parameters via gradient descent, sidestepping the need for previous greedy-search-based or reinforcement learning-based prediction methods in stroke-based image synthesis. Furthermore, we analyze the gradient behaviors of our stroke-based 3D representation and introduce a stroke initialization and update scheme that avoids sub-optimal initialization where optimization is difficult due to local minima.

Our method was evaluated using the multi-view datasets of real-world and synthetic images. Our experiments demonstrate that it effectively creates high-quality artistic 3D scene renderings by maintaining both global visual fidelity and local color accuracy while ensuring perspective consistency. Our contribution is summarized as follows:

- We propose a novel method to translate multi-view 2D images into stylized 3D scenes using 3D strokes based on basic primitives and spline curves whose parameters can be learned through gradient descent.
- We present a novel initialization scheme for the parameters in our stroke-based representation to address issues of flat gradients during optimization.
- Our method supports various 3D stroke styles and can be applied for sophisticated geometry and appearance stylizations similar to those created by human painters.

2. Related Work

2.1. Image Painting

Image painting, evolving over a long history, uses vectorized strokes in various colors and styles for 2D image creation. Early methods like "Paint by Numbers" [9] introduced brush parameters such as position, color, size, and direction, optimized through techniques like stroke zooming, noise adding, and color enhancement. Subsequently, more expressive forms emerged, employing slen-

der spline curves [10] and rectangular strokes [29] to better capture source image details. The development of brush libraries [26, 33] allowed for more diverse styles in paintings, notably in oil painting effects. With the advent of deep learning, techniques like generative neural networks [21, 30] have further refined styled strokes. Advanced loss based on optimal transportation [34] and feed-forward paint transformer [16] are used to improve fidelity and speed up the painting process. While these techniques are relatively mature, applying styles directly to 2D images from 3D scenes lacks multi-view consistency, which motivates our present work on using 3D strokes.

2.2. Stylization of 3D Scenes

The process of 3D scene stylization involves applying the visual style of a chosen reference to a target 3D scene while preserving its inherent 3D structure. Traditional methods [27] faced challenges like limited control over perceptual factors and the inability to selectively target stylization effects. The advent of Neural Radiance Fields (NeRF) has provided a more flexible representation for 3D scene stylization. Techniques like ARF [32] transfer artistic features from 2D style images to 3D scenes, treating stylization as an optimization problem within the NeRF framework. Chiang et al. [3] combines NeRF's implicit representation with a hypernetwork for style transfer. StylizedNeRF [11] jointly learns 2D and 3D stylization, and StyleRF [15] learns high-level features in 3D space for fast zero-shot style transfer. Our approach diverges from existing methods, which rely on additional style references and fundamentally contrast with the processes human artists use to create artworks. We focus on generating artistic images through 3D brush strokes, eliminating the need for style images. Our method more closely resembles image painting techniques than voxel-level manipulations.

3. Methodology

In this section, we introduce our 3D scene stylization framework with stroke-based representation. Our framework mainly consists of three parts: (1) a collection of 3D strokes based on primitives and spline curves, (2) differentiable rendering and composition of strokes, (3) a stroke initialization and update scheme that stabilizes the training process.

3.1. Overview of Stroke Field

NeRF [19] represents the scene as the density $\sigma(\mathbf{x}) \in \mathbb{R}^+$ and RGB radiance $\mathbf{c}(\mathbf{x}, \mathbf{d})$, modeled by an MLP that takes the spatial coordinates $\mathbf{x} \in \mathbb{R}^3$ and view directions $\mathbf{d} \in \mathbb{R}^2$ as input. Given a camera pose, rays $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ are sampled and cast from the camera's center $o \in \mathbb{R}^3$ along the direction $\mathbf{d} \in \mathbb{R}^3$ passing through each pixel of the image. The color of a ray is given by volume rendering:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} \sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d}) \exp(\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds)dt \quad (1)$$

NeRF

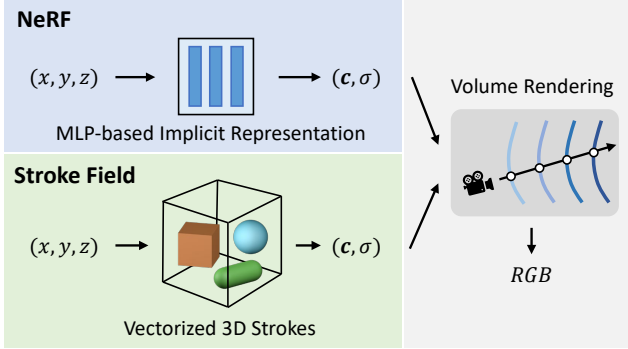$(x, y, z) \longrightarrow$ MLP-based Implicit Representation $\longrightarrow (c, \sigma)$

Stroke Field

$(x, y, z) \longrightarrow$ Vectorized 3D Strokes $\longrightarrow (c, \sigma)$

Volume Rendering

$\downarrow$

RGB

Figure 2. Our method learns a vectorized stroke field instead of MLP-based implicit representation to represent a 3D scene.

Primitive	Params	SDF formula
Sphere	None	$\|\mathbf{p}\|_2 - 1$
Cube	None	$\min(\max(\mathbf{q}_x, \mathbf{q}_y, \mathbf{q}_z), 0) \quad + \quad \|\max(\mathbf{q}, \mathbf{0})\|_2$, where $\mathbf{q} = |\mathbf{p}| - \mathbf{1}$
Tetrahedron	None	$(\max(|\mathbf{p}_x + \mathbf{p}_y| - \mathbf{p}_z, |\mathbf{p}_x + \mathbf{p}_y| + \mathbf{p}_z) - 1)/\sqrt{3}$
Octahedron	None	$(\|\mathbf{p}\|_1 - 1)/\sqrt{3}$
Round Cube	r	$\min(\max(\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z), 0) \quad + \quad \|\max(\mathbf{p}, \mathbf{0})\|_2 - r$
Triprism	h	$\max(|\mathbf{p}_y| - h, \max(|\mathbf{p}_x| * \sqrt{3}/2 + \mathbf{p}_z/2, -\mathbf{p}_z) - 0.5)$
Capsule Line	h, r_δ	$\|\mathbf{p} - [0, \min(\max(\mathbf{p}_y, -h), h), 0]\|_2 - r_\delta \min(\max((0.5*(\mathbf{p}_y+h)/h, 0), 1) - 1$

Table 1. SDF formula for common unit geometry primitives.

As shown in Fig. 2, while NeRF models the scene at per-voxel level with an implicit field, our method represents the scene as a set of vectorized 3D strokes. Similar to NeRF, the stroke field is defined by two spatially varying functions for density $\sigma(\mathbf{x}) \in \mathbb{R}^+$ and RGB color $\mathbf{c}(\mathbf{x}, \mathbf{d})$ and rendered into 2D images at a given camera pose, using the same differentiable volume rendering formula as in NeRF.

While the stroke field shares NeRF's field definition, their core formulations are fundamentally different. Each point's density and color in the stroke field are set by 3D strokes, shaped and styled to resemble brush traces in human drawings. This is analogous to the difference between a rasterized image and a vectorized image but in 3D space. Specifically, the geometry, appearance, and opacity of 3D strokes are given by three parameters, including shape $\theta_{\mathbf{s}} \in \mathbb{R}^{N \times d_s}$, color $\theta_{\mathbf{c}} \in \mathbb{R}^{N \times d_c}$ and density $\theta_\sigma \in (\mathbb{R}^+)^N$, where N is the total number of strokes in the field, d_s and d_c are the number of parameters of the specific stroke shape and color, respectively. We combine these strokes in a differentiable way to acquire the density and color field:

$$(\sigma, \mathbf{c}) = StrokeField(\mathbf{x}, \mathbf{d}; \theta_{\mathbf{s}}, \theta_{\mathbf{c}}, \theta_\sigma) \qquad (2)$$

3.2. 3D Strokes

We first define the shape of our 3D strokes to paint them into the 3D space. The shape of a 3D stroke is essentially a volume region formed by a closed two-dimensional surface. To define such volume, we use the Signed Distance Field (SDF), a scalar value function $\mathrm{sdf}(\mathbf{x}) \to s \in \mathbb{R}$ that gives the signed distance from a point to the closed surface to describe the stroke shape in 3D. The sub-space $\{\mathbf{p} \in \mathbb{R}^3 \mid \mathrm{sdf}(\mathbf{p}) \leq 0\}$ enclosed by the zero level-set is the stroke volume. We construct two types of strokes based on basic geometric primitives and spline curves respectively.

3.2.1 Basic Primitives

Our first category of 3D strokes is based on common geometric primitives such as spheres, cubes, etc. For simplicity, we normalize these geometries at the origin of the unit space, apply a transformation matrix to the unit geometries,

and acquire their SDFs in the scene space. The unit geometry can optionally contain a shape parameter $\theta_s^{\mathrm{basic}} \in \mathbb{R}^{d_{\mathrm{basic}}}$ describing its transformation-agnostic shape feature, where d_{basic} is the number of basic shape parameters. The SDF of unit primitive is defined as $\mathrm{sdf}_{\mathrm{unit}} : (\hat{\mathbf{p}}, \theta_s^{\mathrm{basic}}) \to s \in \mathbb{R}$, where $\hat{\mathbf{p}}$ is a 3D point in the unit space. We list the common unit geometric primitives in Tab. 1.

We apply a transformation matrix $\mathbf{T} \in \mathbb{R}^{4 \times 4}$ to transform primitives from the unit coordinates $\hat{\mathbf{p}}$ to the shared scene coordinates $\mathbf{p} = \mathbf{T}\hat{\mathbf{p}}$. The parameters of transformation are composed of a translation $\mathbf{t} \in \mathbb{R}^3$, a rotation described by Euler angle $\mathbf{r} \in \mathbb{R}^3$, and a uniform or anisotropic scale $\mathbf{s} \in \mathbb{R}^3$. To acquire the SDF of primitives in the scene space, we inversely transform the position to the unit space and query the unit SDF:

$$\mathrm{sdf}(\mathbf{p}; \theta_s) = \mathrm{sdf}_{\mathrm{unit}}(\mathbf{T}(\mathbf{t}, \mathbf{r}, \mathbf{s})^{-1}\mathbf{p}, \theta_s^{\mathrm{basic}}), \qquad (3)$$

where $\theta_s = \{\theta_s^{\mathrm{basic}}, \mathbf{t}, \mathbf{r}, \mathbf{s}\}$ is the final shape parameters for primitive-based 3D strokes. In practice, we may use a subset of translation, rotation, and scale to combine different styles of primitive strokes. We list all the primitive-based 3D strokes in the supplementary. More primitive can be easily added given a defined unit SDF.

3.2.2 Spline Curves

In addition to basic primitives, we use volumetric 3D curves with a given radius to simulate the trace of strokes in human paintings. The curves are defined by parametric 3D splines in the scene space: $C : (t, \theta_s^{\mathrm{curve}}) \to \mathbf{x} \in \mathbb{R}^3$, where $t \in [0, 1]$ is the interpolation parameter, and $\theta_s^{\mathrm{curve}} \in \mathbb{R}^{d_c}$ is the parameter of the 3D spline. To simulate the brushstroke effects, we define two different radii $r_a, r_b \in \mathbb{R}^+$ at the two endpoints of the spline curve respectively, and interpolate the radius at any position on the curve smoothly based on t as $r(t; r_a, r_b) = r_a(1 - t) + r_b t$.

We utilize common polynomial splines, including quadratic and cubic Bézier and Catmull-Rom splines, to define 3D curves. To compute the SDF of a 3D curve, we need

to solve for the value of t corresponding to the nearest point on the spline curve for any point $\mathbf{p} \in \mathbb{R}^3$ in space. While there exists an analytic solution for quadratic splines, it is difficult to directly solve for the nearest point's t value for cubic or more complex splines in a differentiable way.

Therefore, we use a general approximation method to compute the SDF: we uniformly sample $K + 1$ positions on the curve to get K line segments, and calculate the distance from these segments to the query point respectively, thereby obtaining an approximate nearest point. This algorithm can be applied to any parametric curve spline and allows simple control over the trade-off between computational complexity and accuracy. With the found t value of the nearest point on the 3D spline to query position $\mathbf{p}$ as t^*, the SDF of the 3D curve is defined as

$$\text{sdf}(\mathbf{p}; \theta_s) = \|\mathbf{p} - C(t^*, \theta_s^{\text{curve}})\|_2 - r(t^*; r_a, r_b) \quad (4)$$

where $\theta_s = \{\theta_s^{\text{curve}}, r_a, r_b\}$ is the final shape parameters for spline-based 3D strokes. We leave the details of the nearest point finding algorithm in the supplementary.

3.3. Differentiable Rendering of 3D Strokes

With the SDF of 3D strokes defined in Sec. 3.2, we now convert it into the density field $\sigma(\mathbf{x})$ and color field $\mathbf{c}(\mathbf{x}, \mathbf{d})$ for differentiable rendering.

Deriving Density Field from SDF. Theoretically, we consider the inner region of a 3D stroke with an SDF value less than or equal to zero. Therefore, we can define an indicator region function $\alpha(\mathbf{x}) \in [0, 1]$ based on whether a point is inside the SDF:

$$\alpha(\mathbf{x}) = \begin{cases} 1, & \text{sdf}(\mathbf{x}) \leq 0 \\ 0, & otherwise \end{cases} \quad (5)$$

Assuming uniform density inside one stroke, we define the density field of each 3D stroke as

$$\sigma(\mathbf{x}) = \theta_\sigma \alpha(\mathbf{x}) \quad (6)$$

where $\theta_\sigma \in \mathbb{R}^+$ is the density parameter of a 3D stroke. However, due to the discontinuity of the indicator function, the gradient of the density field w.r.t. the shape parameters of the SDF is zero, thus hindering the use of losses to optimize the 3D stroke. To render the 3D stroke in a differentiable way, we aim to derive a smooth approximation of the region function to gain non-zero gradients regarding the shape parameters. We use the Laplace cumulative distribution function (CDF) to approximate the discrete region function, similar to VolSDF [31]:

$$\alpha(\mathbf{x}) = \begin{cases} 1 - \frac{1}{2} \exp(\text{sdf}(\mathbf{x})/\delta), & \text{sdf}(\mathbf{x}) \leq 0 \\ \frac{1}{2} \exp(-\text{sdf}(\mathbf{x})/\delta), & otherwise \end{cases} \quad (7)$$

where δ controls the width of the smooth transitional interval, as shown in Fig. 3. This definition ensures that all
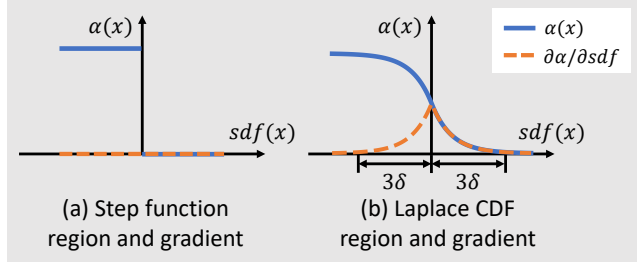

Figure 3. Differential region function by approximating step function with the CDF of Laplace distribution. The parameter δ controls the width of the transitional area with respect to the SDF of 3D strokes, where a larger δ leads to smoother gradient at the cost of making the shape boundary blurrier.

points in space have non-zero derivatives w.r.t. the shape parameters, although the gradient would be too small to achieve meaningful optimization excluding the near region besides the zero level-set boundary of the SDF. This approximation approaches the discrete indicator function when $\delta \to 0$, while larger δ allows smooth gradients flow in a larger area near the boundary, at the cost of making the shape boundary more blurred.

Adaptive Choice of δ based on Cone Tracing. Since the pixel corresponding to each ray has a specific planner size during rendering, the pinhole camera model casts a frustum with non-zero volume instead of a ray. The frustum far away from the camera will cover a large region of the scene, while the near frustum will only affect a small region. Using a uniform δ across the entire scene to render the 3D strokes will make the near objects over-blurry, and far objects lack enough gradients to optimize the stroke shape. To solve this issue, we adopt a similar cone tracing strategy as in MipNeRF [1], and adjust the δ value of the region function adaptively based on the cone size. For simplicity, we use isotropic spheres instead of anisotropic multivariate Gaussians to approximate the volume of the cone. Specifically, assuming the radius of each pixel is $\dot{r}$, we calculate the radius of a sphere at sample points as $r = \dot{r}tf^{-1}$, where t is the un-normalized ray distance, and f is the focal length. We then compute the adaptive δ as $\delta = (k_\delta r)^{-1}$, where k_δ is a hyper-parameter controlling how much to "dilate" the boundary. Moreover, as the scale of δ is defined in scene space, for basic primitives with a transformation, we further adjust δ based on the scaling factor of the inverse transform.

Color Field. We mainly use constant color for each stroke, which is irrelevant to the position and view direction and can be expressed as $\mathbf{c}(\mathbf{x}, \mathbf{d}) = \theta_\mathbf{c}$, where $\theta_\mathbf{c} \in \mathbb{R}^3$ is the RGB color parameter. More diverse visual effects can be achieved by spatially varying color fields and joint modification of the density and color. The color field can be readily expanded to support view-dependent effects by replacing the RGB color with spherical harmonics, however,

we have found that the view dependency does not contribute much to the visual effect of stroke-based scenes in practice.

3.4. Composition of 3D Strokes

Given the density and color field of individual 3D strokes, we now compose them into a stroke field for rendering the full 3D scene. In the procedure of image painting, different strokes are painted onto the canvas sequentially, and new strokes are overlayed on the old strokes. This behavior is simulated using the alpha composition. Similarly, we use an "overlay" method to compose multiple 3D strokes.

Suppose we have the density fields $\{\sigma_i(\mathbf{x})\}$ and color fields $\{\mathbf{c}_i(\mathbf{x})\}$ of N strokes, where $i \in \{1, 2, \cdots, N\}$. We use the region function defined in Sec. 3.3 as the blending weight for overlay. Given $\{\alpha_i(\mathbf{x})\}$ as the region function of each 3D stroke, the density and color of the final stroke field are expressed as

$$\sigma(\mathbf{x}) = \sum_i^n \sigma_i(\mathbf{x})\alpha_i(\mathbf{x})T_i(\mathbf{x}),$$
$$\mathbf{c}(\mathbf{x}) = \frac{\sum_i^n \mathbf{c}_i(\mathbf{x})\alpha_i(\mathbf{x})T_i(\mathbf{x})}{1 - T_n(\mathbf{x})}, \tag{8}$$

where

$$T_i(\mathbf{x}) = \prod_j^i (1 - \alpha_j(\mathbf{x})). \tag{9}$$

Note that we normalize $\mathbf{c}(\mathbf{x})$ using the total weight of all region functions, making the final color unaffected by the number of strokes.

The "overlay" composition is sensitive to the painting order of multiple strokes, i.e., the 3D strokes painted later will have larger blending weights than the previously painted strokes. Normally this is the behavior we desire, however, we sometimes would like the final outcome irrelevant to the painting order. The simplest way is choosing the 3D stroke with the maximum $\alpha(\mathbf{x})$ as: $\sigma(\mathbf{x}) = \sigma_i(\mathbf{x})$, $\mathbf{c}(\mathbf{x}) = \mathbf{c}_i(\mathbf{x})$, where $i = \arg\max_j\{\alpha_j(\mathbf{x})\}$. This "max" composition will only attribute the gradient of the density field to the nearest 3D stroke. A smoother way is to use Softmax for computing weights as: $\sigma(\mathbf{x}) = \sigma_i(\mathbf{x})\omega_i(\mathbf{x})$, $\mathbf{c}(\mathbf{x}) = \mathbf{c}_i(\mathbf{x})\omega_i(\mathbf{x})$, where $\omega_i(\mathbf{x}) = \frac{\exp(\alpha_i(\mathbf{x})/\tau)}{\sum_j^n \exp(\alpha_j(\mathbf{x})/\tau)}$, and τ is a hyper-parameter that controls the blending smoothness. We analyze the effect of different compositions in Sec. 4.3.

3.5. Training Strategy

Now we have completed our rendering framework for 3D strokes. We can render 2D images with a given camera pose from parameters of N strokes $\{\theta_s, \theta_c, \theta_\sigma\}_i$, $i \in \{1, 2, \cdots, N\}$, and use gradient descent to optimize these 3D stroke parameters. However, in the previous analysis, we mentioned that the magnitude of gradients vanishes for faraway points from the existing boundary of strokes, thus

leaving shape parameters unable to gain enough optimization. To this end, we propose a training scheme for better stroke initialization and update.

Error Field. When adding 3D strokes to an existing scene, we wish to initialize the new stroke at the location where it is most needed. Therefore, during the training of the stroke field, we additionally train an error field $e(\mathbf{x}) \in \mathbb{R}^+$ to describe the "error rate" at each position of the reconstructed scene, with the region of highest error considered to be the locations most in need of additional 3D strokes. The error value $E(\mathbf{r})$ along each ray $\mathbf{r}$ is obtained by the modified volume rendering formula:

$$E(\mathbf{r}) = \int_{t_n}^{t_f} e(\mathbf{r}(t)) \exp(\int_{t_n}^t e(\mathbf{r}(s))ds)dt. \tag{10}$$

We define the error as the squared difference in color between the stroke field rendering result and the ground truth. Moreover, we prefer the error field to conservatively estimate errors, therefore we use a coefficient $k > 1$ to amplify the loss where the error is underestimated. Assuming the rendering color of stroke field is $C(\mathbf{r})$, and the GT color is $C_{gt}(\mathbf{r})$, then the training loss for the error field is

$$\mathcal{L}_{err} = |d|k^{\max(-sgn(d),0)}, \ d = E(\mathbf{r}) - \|C(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2, \tag{11}$$

where $sgn(\cdot)$ is the sign function. At the same time, for views that are less trained, we apply a regularization loss to the error values. For N total sampling points $\{\mathbf{x}_i\}$ in a batch, the regularization loss is $\mathcal{L}_{err\,reg} = \sum_i^N e(\mathbf{x}_i)$.

Stroke Initialization and Update. When training a scene with N strokes, we randomly initialize N_{start} strokes and then incrementally add the remaining strokes step by step to the scene. We also decay the initial size of strokes based on the current number of strokes, i.e., we progressively use smaller strokes as the number of existing strokes increases. When initializing the i-th stroke, we uniformly sample M coordinates $\{\mathbf{p}_i\}$ within the bounding box of the scene, calculate the error values $\{e_i\}$ in the error field, and select the sampling point with the highest error as the initial position for the 3D stroke. For primitives, we set their translation vector to this initial position. For spline curves, we set their control points to this initial position plus a random offset vector sampled within the current stroke size volume. This initialization scheme, guided by the error field, effectively mitigates optimization issues caused by flat gradients. Additionally, we reinitialize strokes with near-zero density by resampling the current error field. This simple technique recycles some of the brush strokes that might have been trapped in local minima due to poor initialization.

Training Objective. Similar to NeRF, we use multi-view images as the primary supervision. We randomly sample
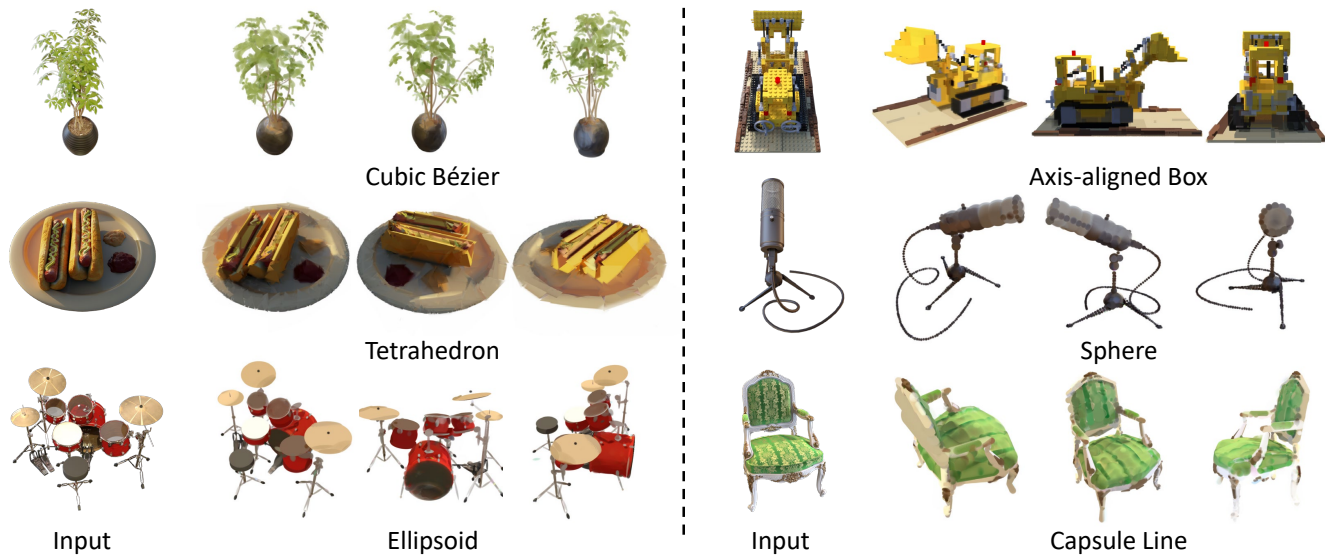
Figure 4. Novel view synthesis results of various 3D strokes on the synthetic scenes. Each scene contains 500 strokes. Our vectorized stroke representation is able to recover the 3D scenes with high fidelity while maintaining a strong geometric style defined by the strokes.

rays from images in the training set and optimize the error between the rendering result of each ray and the ground truth. We use the Charbonnier distance for color loss:

$$\mathcal{L}_{color} = \sqrt{\|C(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2^2 + \epsilon} \qquad (12)$$

For scenes with a ground truth mask, we also impose mask supervision based on the ray's opacity $O(\mathbf{r}) = \exp(\int_{t_n}^{t_f} \sigma(\mathbf{r}(t))dt)$ and the ground truth $M(\mathbf{r}) \in \{0, 1\}$,

$$\mathcal{L}_{mask} = \sqrt{\|O(\mathbf{r}) - M_{gt}(\mathbf{r})\|_2^2 + \epsilon} \qquad (13)$$

Additionally, we regularize the density parameter of N strokes as $\mathcal{L}_{den\,reg} = \sum_i^N |\theta_\sigma^i|$. Combining the above losses, our total loss is given by:

$$\begin{aligned} \mathcal{L} = \ &\lambda_{color}\mathcal{L}_{color} + \lambda_{mask}\mathcal{L}_{mask} \\ &+ \lambda_{den\,reg}\mathcal{L}_{den\,reg} + \lambda_{err}\mathcal{L}_{err} + \lambda_{err\,reg}\mathcal{L}_{err\,reg} \end{aligned} \qquad (14)$$

4. Experiments

We test our method on both synthetic objects from Blender [19] and face-forwarding scenes from LLFF [18]. Each dataset comprises dozens to hundreds of multi-view images with given camera poses. We show our reconstruction results based on various types of vectorized 3D strokes, compare the stylization results of our method with other image-to-painting methods, and conduct an ablation study.

4.1. Stroke-based scene reconstruction

Fig. 4 shows a sample of the stylized 3D synthetic scenes reconstructed by our method with several types of 3D strokes. Our method is able to synthesize 3D scenes based on vectorized 3D strokes, providing strong geometry and appearance

stylization while maintaining the abstracted shapes and colors of the original scenes. Fig. 5 compares reconstruction results of different types of 3D strokes on the same scene. Cubic Bézier and Ellipsoid generally have the best reconstruction fidelity compared to other types of strokes. We list the quantitative metrics of different strokes in the supplementary material. Fig. 6 demonstrates the abstract-to-detail painting results by varying the number of total 3D strokes that are used to reconstruct the scene. We can observe that the lower number of strokes approximates the original scene with a strong shape-related geometric style, while the higher number of strokes leads to reconstruction results with higher fidelity.

4.2. Comparison with other methods

Our method lies in a different category of stylization compared to the common NeRF-based stylization works, which usually take a reference image as input, and transfer the textural style onto the appearance of a trained radiance field. Our method does not require extra reference images, instead, the style of our method is intrinsic and contained in the specification of various 3D strokes.

We compare our method with 2D image-to-painting methods. Fig. 7 shows the vectorized reconstruction results of our method and other 2D vectorized image representation methods, including diffvg [14] and Stylized Neural Painting [34]. We use multi-view images to train our 3D stroke representation, while for 2D methods we select images of the specific viewpoints and apply the vectorization tracing. It can be observed that all these vectorized methods are able to reconstruct original images with obvious stroke styles. Nevertheless, our method can recover more details and strictly maintain the multi-view consistency across dif-
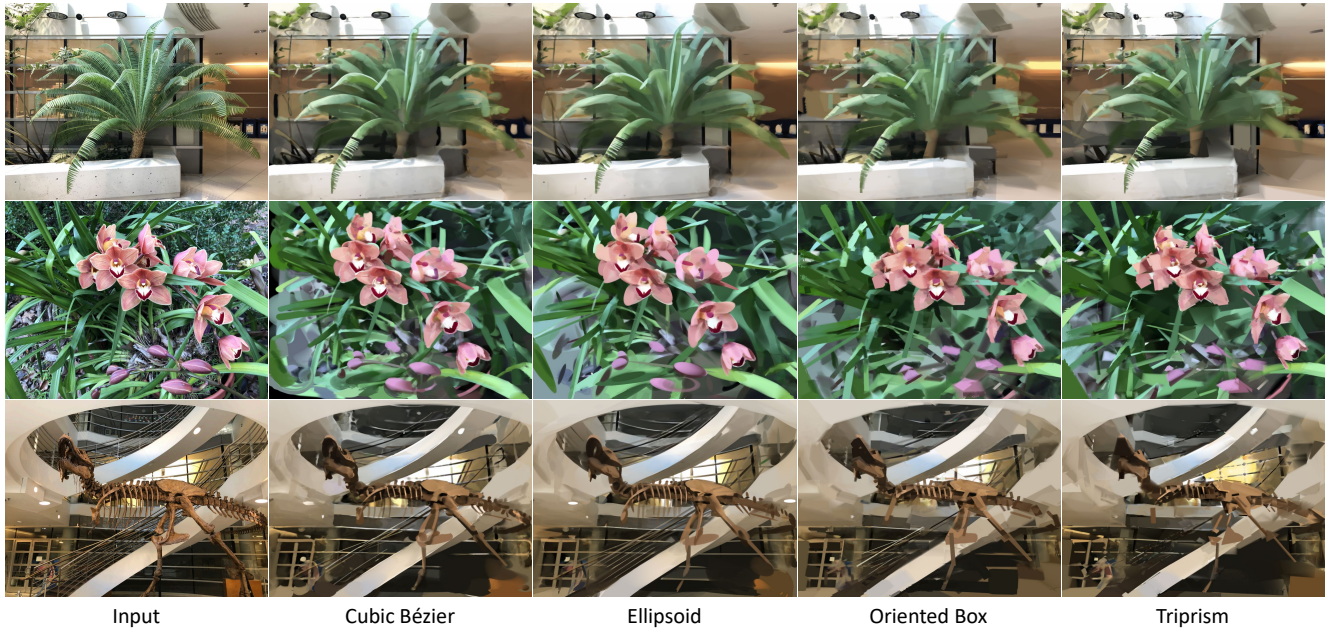
| Input | Cubic Bézier | Ellipsoid | Oriented Box | Triprism |

Figure 5. Results of different types of 3D strokes on the face-forwarding scenes. Each scene contains 1000 strokes.

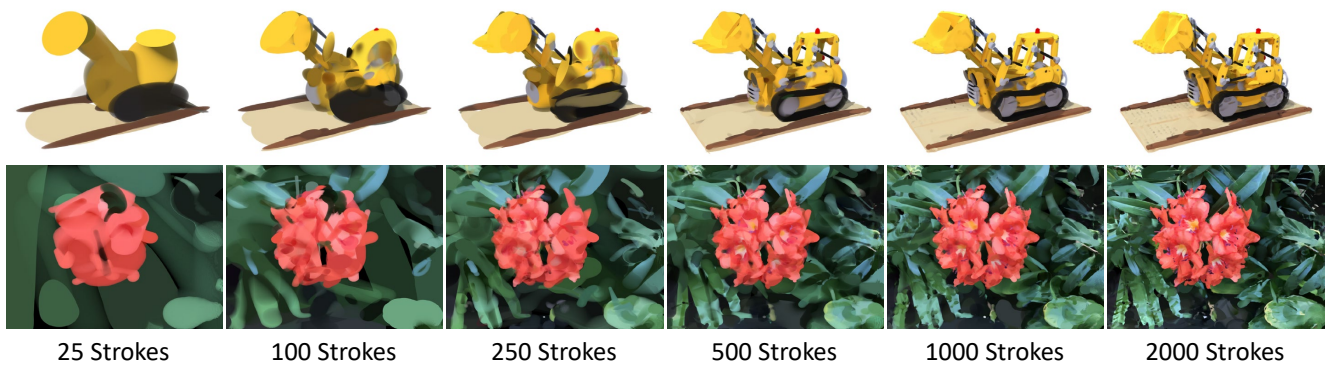| 25 Strokes | 100 Strokes | 250 Strokes | 500 Strokes | 1000 Strokes | 2000 Strokes |

Figure 6. Results with different stroke numbers. By changing the total number of strokes, our stroke-based representation can capture both the abstraction and fine-grained details of the original scenes.
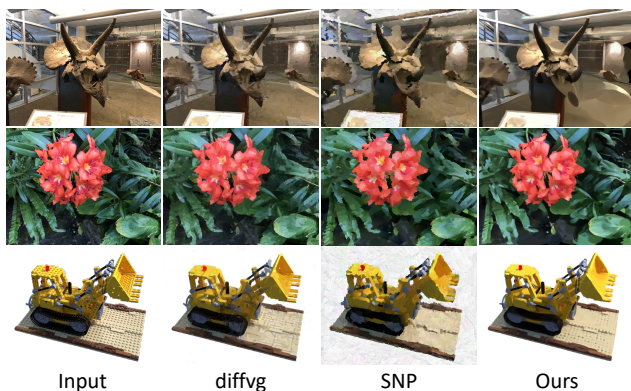

| Input | diffvg | SNP | Ours |

Figure 7. Comparison with 2D image-to-painting methods.

ferent views, which is guaranteed by its 3D representation. The results are best viewed in our supplementary video.

4.3. Ablation studies

Use of adaptive δ in region function. We investigate the effects of employing an adaptive δ in differentiable rendering. The results in Tab. 2 show that omitting adaptive δ leads to reduced performance in novel view synthesis. Additionally, Fig. 8 demonstrates that constant δ values, whether low or high, result in sub-optimal or overly blurry scenes.

Use of error field. In Sec. 3.3, we note that loss gradients are concentrated near the 3D stroke boundary, creating many local minima that heavily influence optimization based on shape initialization. Tab. 2 demonstrates that using the error field for stroke shape initialization significantly improves scene reconstruction fidelity.

Choice of composition function. We compare different composition approaches in Sec. 3.4. We can observe in Fig. 9 that the 'overlay' composition leads to the best re-

Method	PSNR↑	SSIM↑	LPIPS↓
Ours (constant low δ)	21.30	0.793	0.177
Ours (constant high δ)	21.86	0.796	0.235
Ours (w/o adaptive δ)	22.50	0.815	0.179
Ours (w/o error field)	21.79	0.811	0.192
Ours (full)	**23.13**	**0.825**	**0.171**

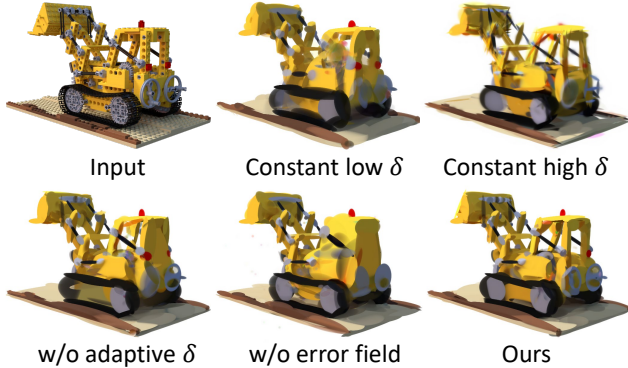Table 2. Novel view synthesis metrics of ablation studies.

Figure 8. Comparison of ablation study results.

Figure 9. Comparison of stroke composition methods.

construction quality, while the 'softmax' composition also reconstructs the scene well. The latter can be chosen if order invariant painting is required.

4.4. Applications

We explore various applications based on our vectorized 3D scene representations, including color transfer and text-driven zero-shot scene drawing.

Color Transfer. As the 3D stroke representation has separate shape and appearance parameters, we can fix the geometry and only fine-tune the color parameter to achieve color transfer effects. We adopt the style loss in common style transfer works, which matches the gram matrix of the feature map outputs from the 4-th and 9-th layers of a pretrained VGG network. The reference style images and color transfer results are demonstrated in Fig. 10.

Text-driven Scene Drawing. We also explore using the vectorized 3D stroke representation to achieve scene creation tasks under a text-guided zero-shot generation framework. We use CLIP [24], a vision-language model that embeds the 2D images and text prompts into the same embedding space. Following DreamField [12] and CLIPDraw [6],

Figure 10. Color transfer results. The left column is the input and style target images respectively.

Figure 11. Text-driven 3D drawings of common objects with 300 strokes of cubic Bézier curve.

we sample camera poses following a circular path around the scene and render an image patch, then optimize the distance between CLIP embeddings of the patch and the text prompt. Fig. 11 shows the generated 3D drawings of different objects. More details are provided in the supplementary.

5. Discussion

We present a novel method to stylize a 3D scene from multi-view 2D images. Different from NeRF-based representations, our method represents the scene as vectorized 3D strokes, mimicking human painting during scene reconstruction process. We demonstrate that this stroke-based representation can successfully stylize 3D scenes with large geometry and appearance transformations, which was not achieved with previous NeRF stylization approaches.

Limitations and future works. Our method uses a stroke setting that demands manual effort to design stroke shapes and appearances. The 3D strokes can be further learned with a generative framework to create a variety of stroke types, like ink and oil brushes, with generated and more detailed SDFs. Moreover, our method may require numerous strokes to represent very complex scenes, partly due to the existence of many local minima during the optimization. Incorporating a globally aware loss, like the optimal transport loss in [34] into 3D space, may enhance the convergence efficiency of our method, which we leave as future work.

References

[1] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 4

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *arXiv preprint arXiv:2304.06706*, 2023. 3

[3] Pei-Ze Chiang, Meng-Shiun Tsai, Hung-Yu Tseng, Wei-Sheng Lai, and Wei-Chen Chiu. Stylizing 3d scene via implicit representation and hypernetwork. In *WACV*, 2022. 1, 2

[4] Tara Chittenden. Tilt brush painting: Chronotopic adventures in a physical-virtual threshold. *Journal of contemporary painting*, 4(2):381–403, 2018. 2

[5] Zhiwen Fan, Yifan Jiang, Peihao Wang, Xinyu Gong, Dejia Xu, and Zhangyang Wang. Unified implicit neural stylization. In *European Conference on Computer Vision*, pages 636–654. Springer, 2022. 1

[6] Kevin Frans, Lisa Soros, and Olaf Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *Advances in Neural Information Processing Systems*, 35:5207–5218, 2022. 8

[7] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015. 1

[8] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 1, 3

[9] Paul Haeberli. Paint by numbers: Abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, 1990. 2

[10] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, 1998. 2

[11] Yi-Hua Huang, Yue He, Yu-Jie Yuan, Yu-Kun Lai, and Lin Gao. Stylizednerf: Consistent 3d scene stylization as stylized nerf via 2d-3d mutual learning. In *CVPR*, 2022. 1, 2

[12] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. 2022. 8

[13] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 694–711. Springer, 2016. 3

[14] 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. 6

[15] Kunhao Liu, Fangneng Zhan, Yiwen Chen, Jiahui Zhang, Yingchen Yu, Abdulmotaleb El Saddik, Shijian Lu, and Eric P Xing. Stylerf: Zero-shot 3d style transfer of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8338–8348, 2023. 1, 2

[16] Songhua Liu, Tianwei Lin, Dongliang He, Fu Li, Ruifeng Deng, Xin Li, Errui Ding, and Hao Wang. Paint transformer: Feed forward neural painting with stroke prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021. 2

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 3

[18] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 6

[19] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2, 6

[20] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 3

[21] Reiichiro Nakano. Neural painters: A learned differentiable constraint for generating brushstroke paintings. *arXiv preprint arXiv:1904.08410*, 2019. 2

[22] Thu Nguyen-Phuoc, Feng Liu, and Lei Xiao. Snerf: Stylized neural implicit representations for 3d scenes. *ACM Trans. Graph.*, 41(4), 2022. 1

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 3

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 8, 3

[25] Eric Risser, Pierre Wilmot, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017. 1

[26] SangHyun Seo, JinWan Park, and KyungHyun Yoon. A painterly rendering based on stroke profile and database. In *CAe*, pages 9–16, 2009. 2

[27] Charles Sheffield. Selecting band combinations from multi spectral data. *Photogrammetric Engineering and Remote Sensing*, 58(6):681–687, 1985. 2

[28] Falong Shen, Shuicheng Yan, and Gang Zeng. Neural style transfer via meta networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8061–8069, 2018. 1

[29] Michio Shiraishi and Yasushi Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 53–58, 2000. 2

[30] Qian Wang, Cai Guo, Hong-Ning Dai, and Ping Li. Stroke-gan painter: Learning to paint artworks using stroke-style generative adversarial networks. *Computational Visual Media*, 9(4):787–806, 2023. 2

[31] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021. 4

[32] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. In *ECCV*, 2022. 1, 2

[33] Mingtian Zhao and Song-Chun Zhu. Customizing painterly rendering styles using stroke processes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on non-photorealistic animation and rendering*, pages 137–146, 2011. 2

[34] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. Stylized neural painting. In *CVPR*, pages 15689–15698, 2021. 2, 6, 8