Volumetric Appearance Stylization with Stylizing Kernel Prediction Network

JIE GUO[*], State Key Lab for Novel Software Technology, Nanjing University
MENGTIAN LI[*], State Key Lab for Novel Software Technology, Nanjing University and Kuaishou Technology
ZIJING ZONG, State Key Lab for Novel Software Technology, Nanjing University
YUNTAO LIU, State Key Lab for Novel Software Technology, Nanjing University
JINGWU HE, State Key Lab for Novel Software Technology, Nanjing University
YANWEN GUO[†], State Key Lab for Novel Software Technology, Nanjing University
LING-QI YAN, University of California, Santa Barbara

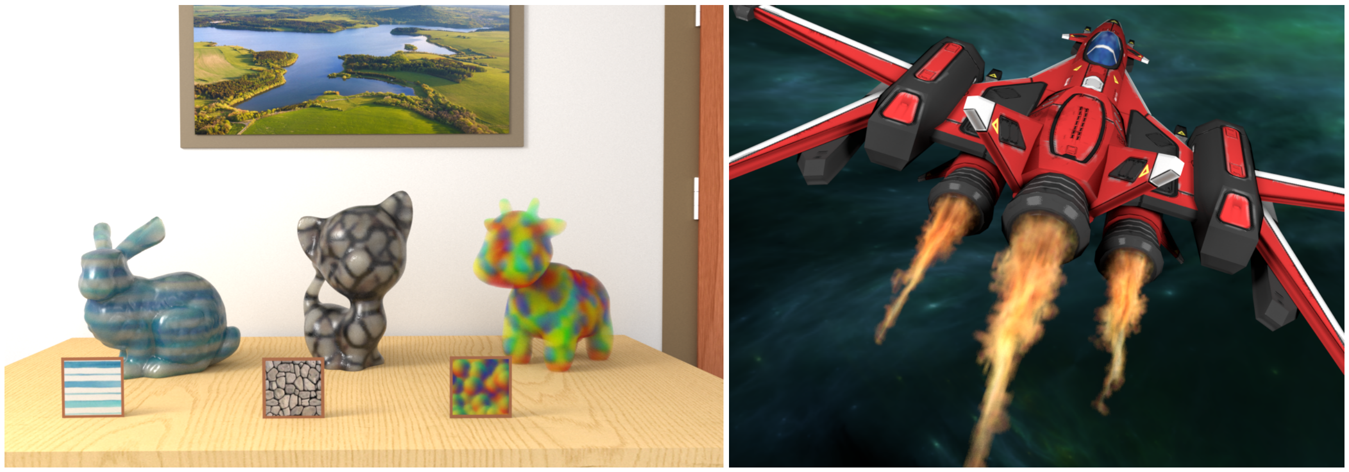

Fig. 1. We propose *stylizing kernel prediction network* (SKPN), a new tool for efficiently stylizing translucent volumes with desired color appearance. Once trained, the proposed SKPN supports arbitrary style transfer and facilitates appearance modelling of many translucent materials, such as different types of marbles (left) and dynamic fires (right, please use Adobe Acrobat and click the rendering to see it animated). Note that the bunny model and the kitten model are slightly darker than the input style images because of an additional dielectric coating on these models.

This paper aims to efficiently construct the volume of heterogeneous single-scattering albedo for a given medium that would lead to desired color appearance. We achieve this goal by formulating it as a volumetric style transfer problem in which an input 3D density volume is stylized using color features extracted from a reference 2D image. Unlike existing algorithms that require cumbersome iterative optimizations, our method leverages a feed-forward deep neural network with multiple well-designed modules. At the core of our network is a stylizing kernel predictor (SKP) that extracts multi-scale feature maps from a 2D style image and predicts a handful of stylizing kernels as a highly non-linear combination of the feature maps. Each group of stylizing kernels represents a specific style. A volume autoencoder (VolAE) is designed and jointly learned with the SKP to transform a density volume to an albedo volume based on these stylizing kernels. Since the autoencoder does not encode any style information, it can generate different albedo volumes with a wide range of appearance once training is completed. Additionally, a hybrid multi-scale loss function is used to learn plausible color features and guarantee temporal coherence for time-evolving volumes. Through comprehensive experiments, we validate the effectiveness of our method and show its superiority by comparing against state-of-the-arts. We show that with our method a novice user can easily create a diverse set of realistic translucent effects for 3D models (either static or dynamic), neglecting any cumbersome process of parameter tuning.

CCS Concepts: • **Computing methodologies** → **Texturing**; *Neural networks.*

[*]Both authors contributed equally to the paper
[†]Corresponding author

Authors' addresses: Jie Guo, State Key Lab for Novel Software Technology, Nanjing University, guojie@nju.edu.cn; Mengtian Li, State Key Lab for Novel Software Technology, Nanjing University and Kuaishou Technology, lemonsky1995@gmail.com; Zijing Zong, State Key Lab for Novel Software Technology, Nanjing University, jimzongzijing@163.com; Yuntao Liu, State Key Lab for Novel Software Technology, Nanjing University, windspectator@gmail.com; Jingwu He, State Key Lab for Novel Software Technology, Nanjing University, hejw005@gmail.com; Yanwen Guo, State Key Lab for Novel Software Technology, Nanjing University, ywguo@nju.edu.cn; Ling-Qi Yan, University of California, Santa Barbara, lingqi@cs.ucsb.edu.

0730-0301/2021/8-ART162 $15.00
https://doi.org/10.1145/3450626.3459799

Additional Key Words and Phrases: Volume rendering, Single-scattering albedo, Stylization, Temporal coherence, Deep learning

ACM Reference Format:
Jie Guo, Mengtian Li, Zijing Zong, Yuntao Liu, Jingwu He, Yanwen Guo, and Ling-Qi Yan. 2021. Volumetric Appearance Stylization with Stylizing Kernel Prediction Network. *ACM Trans. Graph.* 40, 4, Article 162 (August 2021), 15 pages. https://doi.org/10.1145/3450626.3459799

1 INTRODUCTION

Simulating the visual effects of heterogeneous media in Computer Graphics requires carefully fine-tuning a set of material properties, including the extinction coefficient, the single-scattering albedo and the phase function [Cerezo et al. 2005; Schmidt et al. 2016]. Often, the phase function is assumed to be homogeneous and can be set empirically according to some analytical models, e.g., the Henyey-Greenstein [1941] model. The other two properties are expected to vary spatially and their values are typically stored in two volumes (or a volume with multiple channels). Conventionally, adjusting these two volumes to produce a desired appearance is a lengthy task that involves tedious user interference and expensive computation.

To synthesize a density volume that supplies the spatially-varying extinction coefficients in a medium, we often resort to procedural approaches [Ebert et al. 2002] or physically-based fluid simulations [Bridson 2015; Bridson et al. 2006; Fedkiw et al. 2001; Treuille et al. 2003], both of which can achieve controllable behaviors. Despite the difficulty in controllability and the inefficiency in computation, physically-based fluid simulations are preferred in general since they achieve realistic volumes of spatially-varying densities.

As for the single-scattering albedo, fluid simulations do not apply in a straightforward way. Typically, artistic appearance editing boils down to a tedious process of trial and error [Schmidt et al. 2016]. In the 2D image space, changing the appearance via adjusting the albedo only requires manipulating pixel values in local areas, possibly with some underlying physics of light transport [Carroll et al. 2011; Dong et al. 2015]. For 3D volumes, the problem however becomes rather complicated. Due to transparency (or translucency) and stereoscopic effects, modifying the albedo in one view will affect the entire volume in a non-intuitive manner. This makes the direct editing of albedo volumes impractical. Although volumetric reconstruction methods based on multi-view inputs make the problem tractable [Hasinoff and Kutulakos 2007; Ihrke and Magnor 2004; Klehm et al. 2014; Okabe et al. 2015; Shen et al. 2018], they are subject to correct images from multiple viewpoints. Hence, it is more attractive and intuitive to artistically control the appearance of a medium according to only one guided image, which is the goal of this paper.

Specifically, we aim to determine the spatially-varying albedo volume of a heterogeneous medium such that its appearance resembles a source image. This task is very challenging and ill-posed, since the objective has a very high dimensionality. We show that style transfer methods can be adapted to transfer color features extracted from a 2D image to a 3D volume, yielding a similar appearance to the image after physically-based rendering. Unlike style transfer in the 2D image space [Jing et al. 2019], this *volumetric style transfer* (or *volumetric appearance stylization*) problem faces several inherent challenges. First, a proper feature extractor is necessary for extracting useful features from the input image. Second, a similarity merit is required to measure the distance between the input image and the stylized volume. Third, the proposed approach is expected to support multiple styles and can generate temporally coherent animation sequences with high efficiency.

To ameliorate these issues, we resort to a deep learning-based framework, inspired by the recent success of deep learning in image style transfer [Jing et al. 2019]. Fig. 2 gives a high-level overview of our framework. At the core of our framework is *stylizing kernel prediction network* (SKPN), a feed-forward neural network that contains several important components. For feature extraction, we adopt a pre-trained VGG-16 network [Simonyan and Zisserman 2015], augmented with several fully-connected (FC) layers. These layers generate important feature maps from the input style image on multiple scales. To allow arbitrary style transfer, we further compress these feature maps into a group of adaptive convolutional kernels and the affine parameters of density-aware instance normalization (DAIN), termed as stylizing kernels, with the help of several additional FC layers. These stylizing kernels are then convolved with different feature maps in the decoder part of a deep autoencoder [Hinton and Salakhutdinov 2006], enabling the autoencoder to transform a density volume to a stylized albedo volume. To measure the similarity between the stylized albedo volume and the input image, we propose a differentiable *volume rendering layer* to project the volume into the 2D image space and compute the difference based on the color statistics in both the feature space and the RGB space. A temporal loss is introduced to enforce temporal coherence between adjacent frames for time-evolving volumes. Since all our components are differentiable, the full network can be trained end-to-end. With our careful design, the proposed framework has the generalization ability to new styles at the inference stage and can efficiently handle different density volumes, without encoding the style directly in the autoencoder.

To the best of our knowledge, our method is the first to leverage deep neural networks to generate a high-dimensional albedo volume given only one input image. Some graphical applications will benefit from our method. For instance, our method facilitates hallucinating the appearance of a translucent object according to a single real image [Song et al. 2009]. It can also be an efficient and intuitive tool for texturing 3D flows (e.g., fires) with consistent and stable stylized animations.

In summary, we make the following contributions:

- We propose a novel framework for transferring color features from 2D images to 3D volumes, yielding similar visual appearance.
- We design a multi-scale kernel-based neural network to support arbitrary style transfer and ensure temporal coherence for animated volumes.
- We introduce a density-aware instance normalization layer to avoid color drift easily incurred by very sparse but condensed density volumes.
- We implement an analytical differentiable volume rendering layer to convert an albedo volume and its corresponding density volume to 2D images, facilitating the computation of the loss function.

2 RELATED WORK

2.1 Neural Style Transfer for Images

Image style transfer is a long-standing problem that seeks to migrate the style of a reference style image onto another input image. The pioneer work of Gatys et al. [2016] first studied the way of using a CNN to reproduce famous painting styles on natural images. Since then, a surge of follow-up studies are conducted to improve the performance, opening up a new field named *neural style transfer* (NST) [Jing et al. 2019]. Existing NST algorithms are either image-optimization-based or model-optimization-based. The former category transfers styles by iteratively optimizing an image, possibly with summary statistics [Gatys et al. 2016; Li et al. 2017b; Luan et al. 2017; Risser et al. 2017; Ruder et al. 2018], while the second optimizes a neural network offline and generates stylized images with a single forward pass [Chen et al. 2017; Chen and Schmidt 2016; Cheng et al. 2020; Li et al. 2017a, 2018b; Shen et al. 2018]. Currently, model-optimization-based offline neural methods are preferred since they can support arbitrary style transfer once trained. Considering this, we also explore model-optimization-based strategy to implement our SKPN, allowing it to stylize volumes in a single forward pass. For a comprehensive review of NST, please refer to [Jing et al. 2019].

2.2 Neural Style Transfer for 3D Contents

Nowadays, 3D contents are becoming widely available and easier to capture [Park et al. 2018]. This brings about an increasing interest in stylizing 3D representations. Several recent methods try to transfer the style of an image onto a 3D mesh, facilitating the editing of 3D surfaces [Kato et al. 2018; Liu et al. 2018]. Closer to ours are those works focusing on stylizing volumetric data either by optimization-based texture synthesis [Sato et al. 2018] or by deep neural networks [Chu and Thuerey 2017; Kim et al. 2019, 2020; Xie et al. 2018]. Kim et al. [2019] proposed TNST to transfer semantic structures given by 2D image to 3D fluid simulations, achieving complex artistic patterns. Later, they reformulated TNST in a Lagrangian setting (i.e., LNST) to ensure better temporal consistency and support color information [Kim et al. 2020]. These two methods focus on stylizing shapes of fluid simulations and require a long computational time per frame. Recently, TNST has been extended to enable color transferring [Christen et al. 2020] with another lengthy optimization method. In this paper, we aim to stylize physically-plausible color appearance which has a complex relationship with intrinsic material parameters. Due to the feed-forward nature of the proposed network, high-efficiency stylization is allowed for our method.

2.3 Artistic Appearance Design

Artistic design of appearance in Computer Graphics has attracted considerable attention in both industry and research community [Schmidt et al. 2016]. To mimic a desired appearance, a straightforward strategy is to directly edit the parameters of an underlying material model. However, this strategy is non-intuitive since the final appearance can be neither linear nor low-order polynomial with respect to the parameters. In the field of volumetric appearance design, Song et al. [2009] proposed a new representation that is amenable to various simple parametric and image-based editing

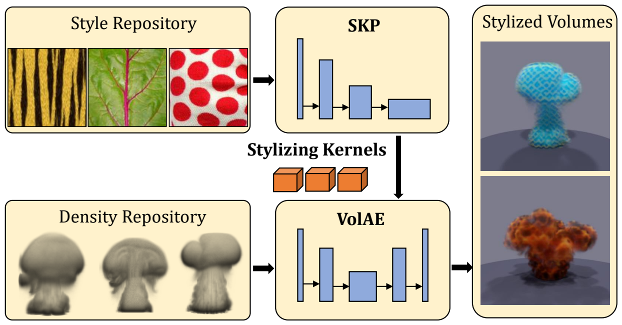

Fig. 2. High-level overview of the proposed framework.

operations for heterogeneous subsurface scattering. Dobashi et al. [2012] used genetic algorithms to search for optimal parameters to create realistic clouds from input images. Hašan and Ramamoorthi [2013] built a material designer to interactively set the single-scattering albedo coefficients, heavily relying on precomputation. Volumetric reconstruction based on multi-view images is also a possible way to reach a desired appearance [Hasinoff and Kutulakos 2007; Ihrke and Magnor 2004; Klehm et al. 2014; Okabe et al. 2015; Shen et al. 2018]. However, preparing correct multi-view inputs is difficult and expensive. There are also some researches focusing on editing the appearance of fluids, ensuring temporal consistency [Bargteil et al. 2006; Gagnon et al. 2016; Kwatra et al. 2007]. Currently, these methods are all based on optimization. Therefore, either cumbersome precomputation or strong constraints (e.g., correct multi-view images) are required to ease the computation in optimization. In comparison, our SKPN is free of these and predicts the desired appearance efficiently.

2.4 Differentiable Rendering

Our work is also closely related to differentiable rendering, the process of computing the derivatives of image pixels with respect to differential changes of virtual scenes. Some approximate differentiable renderers rely on smooth rasterization and ignore global light transport effects [Kato et al. 2018; Liu et al. 2019; Loper and Black 2014; Petersen et al. 2019]. Recently, there is an increasing interest in making physically-based renderers differentiable [Bangaru et al. 2020; Li et al. 2018a; Loubet et al. 2019; Nimier-David et al. 2019; Zhang et al. 2020, 2019]. One line of work leverages boundary sampling to explicitly integrate over the discontinuities along object silhouettes [Li et al. 2018a; Zhang et al. 2019]. Others try to convert the integral over the object silhouette to an area integral by either reparameterization [Loubet et al. 2019] or the divergence theorem [Bangaru et al. 2020]. Due to the inherent complexities, these physics-based differentiable renderers are still too slow for training deep neural networks. In this paper, we propose a lightweight and efficient differentiable volume rendering layer that is tailored for the task of volumetric style transfer.

3 OVERVIEW

We first briefly introduce our framework for volumetric appearance stylization. An overall pipeline is sketched in Fig. 2. The basic structure of the proposed framework is a deep neural network that takes a density volume and a reference style image as input, and outputs a stylized albedo volume in a sense that its style is "similar" to the input image. This network, i.e., *stylizing kernel prediction network* (SKPN), comprises two subnetworks: a *stylizing kernel predictor* (SKP) that extracts color features from an input style image as a group of stylizing kernels, and a *volume autoencoder* (VolAE) that transforms an achromatic density volume to an albedo volume with the help of the stylizing kernels.

After training end-to-end with a density volume dataset and a style image dataset, our SKPN is able to produce a wide range of appearance according to different style images, thanks to our special design of the stylizing kernels. Unlike some previous networks that encode each style image directly in a specified neural network [Johnson et al. 2016; Li and Wand 2016; Ulyanov et al. 2016a], we generate a group of stylizing kernels for a given style image using SKP, a specially designed kernel prediction network. Previous kernel prediction networks [Bako et al. 2017; Chen et al. 2017; Mildenhall et al. 2018; Vogels et al. 2018] usually produce kernels on a single layer of feature maps. In SKP, we generate a group of stylizing kernels from several different layers. Such a design of multi-scale kernels helps to adapt to image structures with different properties. These stylizing kernels are decoupled with the VolAE, avoiding retraining VolAE for every new style. We feed these kernels to the decoder part of VolAE to generate an albedo volume. With this albedo volume and the input density volume supplying extinction coefficients, we are able to render the corresponding participating medium with desired appearance under arbitrary viewpoints. The rendered images are expected to guarantee temporal coherence when changing the viewpoint.

In our framework, SKP and VolAE are jointly trained with two datasets. One dataset contains 3000 style images selected from the DTD dataset [Cimpoi et al. 2014], while the other dataset contains 3000 volumetric densities simulated by *mantaflow* [Thuerey and Pfaff 2018]. We show by experiments that our framework supports arbitrary styles and densities once trained.

4 VOLUMETRIC APPEARANCE STYLIZATION

In this section, we first formulate the problem of volumetric appearance stylization. Then, we unfold the network architecture of the proposed SKPN, including two subnetworks: SKP and VolAE. After that, we introduce a lightweight volume rendering layer to quickly convert a stylized volume into 2D rendered images. Lastly, we describe the loss function and datasets used to train our SKPN.

4.1 Problem Formulation

Since no ground-truth pairs are supplied for training, the volumetric appearance stylization problem can be viewed as an unsupervised learning problem. Specifically, we have a density volume dataset $\mathcal{D}_{\mathbf{V}_\sigma} = \{\mathbf{V}_\sigma^{(1)}, \mathbf{V}_\sigma^{(2)}, ..., \mathbf{V}_\sigma^{(K_\sigma)}\}$ and a style image dataset $\mathcal{D}_{\mathbf{I}_s} = \{\mathbf{I}_s^{(1)}, \mathbf{I}_s^{(2)}, ..., \mathbf{I}_s^{(K_s)}\}$, with K_σ and K_s denoting the example numbers of these two datasets, respectively. Our goal is to find a parameter vector θ minimizing the following objective:

$$\epsilon(\theta) = \sum_{k_1=1}^{K_\sigma} \sum_{k_2=1}^{K_s} \mathcal{L}(\Phi_\theta(\mathbf{V}_\sigma^{(k_1)}, \mathbf{I}_s^{(k_2)}), \mathbf{I}_s^{(k_2)}) \tag{1}$$

where Φ_θ is our network with parameters θ and $\mathcal{L}$ is the loss function. Since no ground-truth albedo volumes exist in our volumetric appearance stylization problem, the loss function $\mathcal{L}$ becomes crucial in our framework. In particular, optimizing in the 3D space faces more instability and inconsistency.

Once trained, our network is expected to quickly generate a stylized albedo volume $\mathbf{V}_\alpha$ from a given density volume $\mathbf{V}_\sigma$ and a specified style image $\mathbf{I}_s$, i.e.,

$$\mathbf{V}_\alpha = \Phi_\theta(\mathbf{V}_\sigma, \mathbf{I}_s). \tag{2}$$

Ideally, the estimation $\mathbf{V}_\alpha$ should have the same appearance with the style image after rendering from any viewpoint and possesses plausible stereoscopic effects.

4.2 Network Architecture

To allow arbitrary style transfer, kernel-based strategies, e.g., StyleBank [Chen et al. 2017], prevail in the field of image style transfer. However, StyleBank uses a single kernel to represent different styles. This is suboptimal for our volumetric style transfer problem since the style image in our problem is 2D while the output is 3D. A single kernel would lose many details due to the mismatch between different spaces. Moreover, StyleBank only pre-trains multiple styles with a fixed number of layers while new image styles are learned incrementally. In contrast, we leverage adaptive kernels to encode arbitrary styles in a feed-forward network.

To preserve fine details as much as possible when transforming from the 2D image space to the 3D space, we propose a multi-scale kernel-based neural network. Such a hierarchical design can generate different scales of style elements better since each hierarchical level contains image formation of a specific level. As shown in Fig. 3, the overall network architecture of the proposed SKPN contains two subnetworks: a SKP that produces a series of stylizing kernels and a VolAE that transforms an input density volume to an albedo volume.

4.2.1 Stylizing kernel predictor (SKP). In SKP, we employ a VGG-16 (pretrained on the ImageNet dataset [Deng et al. 2009]) to extract feature maps from an input style image, as shown in the top half of Fig. 3. Previous networks usually use statistical measurements of VGG-16 feature maps to represent styles, such as the Gram matrix [Chen et al. 2017; Gatys et al. 2016], the histogram of feature activations [Risser et al. 2017] and the mean-variance representation [Huang and Belongie 2017; Li et al. 2017b]. We currently choose the channel-wise mean-variance representation, since the first two will yield a too high-dimension style feature, increasing the size of the network and making training difficult.

As highlighted in the dashed red box of Fig. 3, we extract mean μ and standard deviation (the square root of the variance) σ from the feature maps $\{l_s\}$ in VGG-16 layers and concatenate them into two vectors:

$$\mathcal{V}_\mu^l = \oplus_c\{\mu(\mathcal{F}_c^l(\mathbf{I}_s))\} \tag{3}$$

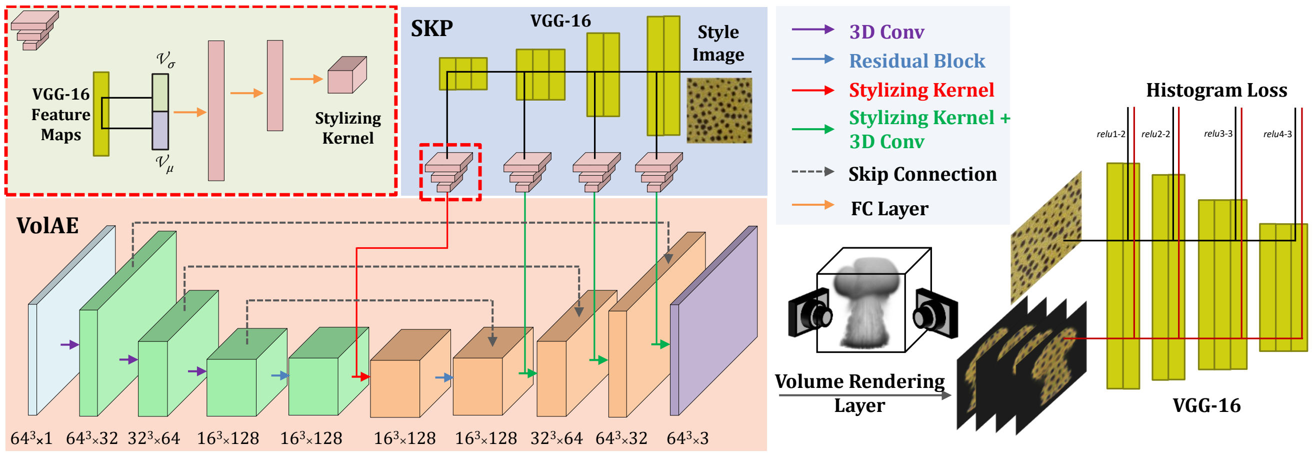

Fig. 3. Network architecture of the proposed SKPN. SKP aims to extract multi-scale stylizing kernels from an input style image while VolAE is responsible for generating a proper albedo volume for an input density volume with the help of these stylizing kernels. In the dashed red box, we show the formulation of the stylizing kernel based on the mean and variance of VGG-16 feature maps.

and

$$\mathcal{V}_\sigma^l = \oplus_c\{\sigma(\mathcal{F}_c^l(\mathbf{I}_s))\} \tag{4}$$

respectively. Here, $\mathbf{I}_s$ is the input style image, $\mathcal{F}_c^l$ is the c-th feature map channel at layer l ($\in \{l_s\}$) of VGG-16 network, and $\oplus$ denotes the concatenation operation. Then, we map each concatenated mean-variance vector $\mathcal{V}_\mu^l \oplus \mathcal{V}_\sigma^l$ to a stylizing kernel $\mathcal{K}^l$ through a series of fully-connected (FC) layers. The input and output feature dimensions of these FC layers are $(|\mathcal{V}_\mu^l \oplus \mathcal{V}_\sigma^l|, S_{\text{FC}})$, $(S_{\text{FC}}, S_{\text{FC}})$ and $(S_{\text{FC}}, C_{\text{in}} \cdot C_{\text{out}} \cdot S_{\text{kernel}}^3 + 3C_{\text{out}})$, respectively. Here, C_{in} and C_{out} denote the channel numbers of input and output feature maps, respectively. $S_{\text{kernel}} = 1$ and $S_{\text{FC}} = 256$ denote the kernel size and the latent vector's dimensionality, respectively. These FC layers are activated by LeakyReLU [Maas et al. 2013]. Such a design of SKP is the key to our framework, as it allows to achieve stylized volumes with arbitrary style images.

We choose four levels of stylizing kernels in our current design, considering training difficulty and GPU memory. Basically, employing one more kernel will incur millions of additional tunable parameters for our network, increasing the risk of overfitting.

4.2.2 Volume autoencoder (VolAE). The encoder and decoder in VolAE consist of two symmetrical architectures, both of which are made of three convolutional blocks. In the encoder, the kernel size of each convolutional layer is $3 \times 3 \times 3$, while the stride is 1 for the first one and 2 for the last two, achieving a $4\times$ downsampling rate. Symmetrically, a $2\times$ tri-linear upsampling layer is in front of the corresponding convolutional layers of the decoder to restore the original resolution. The residual block follows the structure in [He et al. 2016] except replacing 2D convolutions with 3D convolutions. Skip connections are also used between mirrored blocks in the encoder and decoder stacks to preserve local and low-level information from input data as much as possible. The input of VolAE is a density volume which has only one channel while the output is an albedo volume containing three RGB channels. The other channels of feature maps are symmetrical in the encoder and decoder, which are specified in Fig. 3. We apply LeakyReLU [Maas et al. 2013] and ReLU as activations respectively in the encoder and the decoder.

To generate albedo volumes with different styles, we insert 4 stylizing kernels generated by SKP into VolAE. One is between the encoder and the decoder, and the others are in front of the last 3 convolutional layers in the decoder. Each group of stylizing kernels represents a specified style from an input style image.

4.2.3 Density-aware instance normalization. All convolutional layers in VolAE are normalized by *density-aware instance normalization* (DAIN) except the last layer which is fed to a tanh function to guarantee the finite range of output. We use this new normalization layer instead of the standard instance normalization (IN) [Ulyanov et al. 2016b] because the distribution of density usually varies considerably in the process of smoke simulation. Occasionally, a 3D volume will contain a limited number of non-zero voxels that will seriously influence the computation of the mean and variance used in the standard IN. These sparse but condensed density volumes will make training difficult and incur unpleasant color drift, as shown in the first row of Fig. 4. To alleviate this issue, we propose DAIN to let the network pay more attention to those voxels with higher density values while ignoring those zero-valued voxels.

Our DAIN works as follows. We first obtain a smoothing density mask with $\mathbf{m} = 1 - \exp(-\lambda \mathbf{V}_\sigma^2)$, and downsample it with max pooling to fit different resolutions. Currently, we set λ to 100. Then, the mean and variance are computed as

$$\mu_c = \frac{1}{\sum_{i,j,k} \mathbf{m}_{i,j,k}} \sum_i^D \sum_j^H \sum_k^W \mathcal{F}_{c,i,j,k} \cdot \mathbf{m}_{i,j,k} \tag{5}$$

$$\sigma_c^2 = \frac{1}{\sum_{i,j,k} \mathbf{m}_{i,j,k}} \sum_i^D \sum_j^H \sum_k^W (\mathcal{F}_{c,i,j,k} - \mu_c)^2 \cdot \mathbf{m}_{i,j,k} \tag{6}$$

where $\mathcal{F}$ is the feature map fed into DAIN, c refers to the c-th channel of $\mathcal{F}$, and D, H, W are the resolutions of $\mathcal{F}$. With the reweighted mean and variance, our DAIN normalizes the feature map

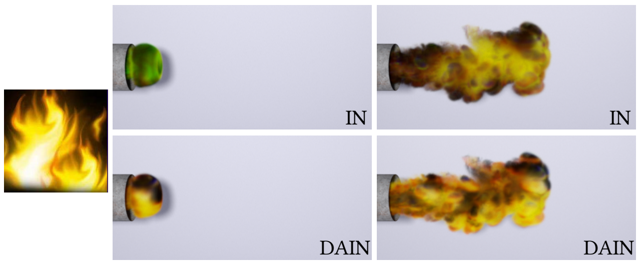

Fig. 4. The effect of DAIN. Compared with the standard IN, our DAIN avoids the problem of color drift, especially when the density volume is very sparse. Here we show two density volumes with different sparsity.

and transforms it with learnable affine parameters γ_c and β_c to

$$\frac{\mathcal{F}_c - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}\gamma_c + \beta_c \tag{7}$$

where ϵ is a small value to avoid division by zero (10^{-5} in our implementation). With this new normalization layer, we are able to preserve the color appearance of the input style for density volumes with diverse sparsity, as shown in the second row of Fig. 4. In comparison, the standard IN yields plausible color appearance when the density volume is relatively dense (the third column), but it fails for very sparse volume (the second column). Since DAIN is general-purpose, we believe it will cater other applications involving 3D volumes.

4.3 Volume Rendering Layer

To measure the similarity between the output volume and the input style image, we develop a lightweight *volume rendering layer* to project the 3D volume into the 2D image space. This allows the measurement to be realized with any image-space method. The basic idea of our volume rendering layer is to synthesize 2D images using a simplified but differentiable volume rendering method. This method takes the original density volume and the stylized albedo volume as the input and quickly generates 2D images with proper scene configurations.

4.3.1 Simplified volume rendering method. Our simplified volume rendering method is based on the following assumptions:

- The medium has an isotropic phase function and is lit by a constant environmental light.
- Only one ray is traced per pixel whose radiance is computed according to single scattering.

Under these assumptions and given the extinction coefficient σ_t (stored in a density volume $\mathbf{V}_\sigma$) and the albedo α (stored in an albedo volume $\mathbf{V}_\alpha$) that both vary with position $\mathbf{x}$, the volume rendering layer is designed to evaluate

$$L(\mathbf{x}, \boldsymbol{\omega}) = \int_0^t Tr(\mathbf{x}_t \to \mathbf{x})\sigma_t(\mathbf{x}_t)\alpha(\mathbf{x}_t)L_s(\mathbf{x}_t)\mathrm{d}t \tag{8}$$

in the forward pass. Here, $L(\mathbf{x}, \boldsymbol{\omega})$ is the radiance arriving at $\mathbf{x}$ along the direction $\boldsymbol{\omega}$ and $L_s(\mathbf{x}_t)$ represents the in-scattered radiance of light. $Tr(\mathbf{x}_t \to \mathbf{x}) = \exp\{-\int_0^t \sigma_t(\mathbf{x}_{t'})\mathrm{d}t'\}$ is the transmittance

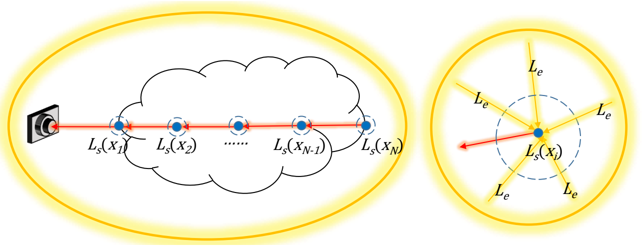

Fig. 5. Illustration of ray marching in our volume rendering layer. Ray marching generates equal-distance samples along each ray lit by a constant environmental light (left). The in-scattered radiance of each sample is determined by a handful of directional samples drawn from an isotropic phase function (right).

between $\mathbf{x}_t$ and $\mathbf{x}$. With ray marching method as depicted in Fig. 5, Eq. (8) is rewritten into a Riemann sum as

$$L(\mathbf{x}, \boldsymbol{\omega}) = \sum_{i=1}^{N} \exp\left\{-\sum_{j=1}^{i}\sigma_t(\mathbf{x}_j)\Delta t\right\}\sigma_t(\mathbf{x}_i)\alpha(\mathbf{x}_i)L_s(\mathbf{x}_i)\Delta t \tag{9}$$

in which N is the number of equal-distance samples along the ray and $\Delta t = t/N$. To compute the in-scattered radiance L_s, we sample K directions using Hammersley sampling on the unit sphere to generate incident rays. Then, we approximate L_s at $\mathbf{x}_i$ as

$$L_s(\mathbf{x}_i) = \frac{L_e}{K}\sum_{k=1}^{K} Tr(\mathbf{x}_{b,k} \to \mathbf{x}_i) \tag{10}$$

where L_e is the radiance of the constant environmental light and $\mathbf{x}_{b,k}$ represents the position on the boundary of the volume along the k-th incident ray. To accelerate the computation, L_s can be pre-computed and stored in a grid. To make the renderings smooth, we perform tri-linear interpolation to sample σ_t, α and L_s for each ray. In our current implementation, we sample $K = 128$ directions and set ray marching steps N to 64 for both L and L_s.

4.3.2 Derivative of volume rendering layer. To allow backward propagation of the gradient, we need to evaluate the partial derivative of the albedo volume $\mathbf{V}_\alpha$. Since the Riemann sum in Eq. (9) is linearly proportional with respect to $\mathbf{V}_\alpha(\mathbf{x}_i)$, the partial derivative can be easily calculated as

$$\frac{\partial L(\mathbf{x}, \boldsymbol{\omega})}{\partial \alpha(\mathbf{x}_i)} = \exp\left\{-\sum_{j=1}^{i}\sigma_t(\mathbf{x}_j)\Delta t\right\}\sigma_t(\mathbf{x}_i)L_s(\mathbf{x}_i)\Delta t. \tag{11}$$

To compute this partial derivative, we also adopt ray marching, in a similar way as that in the forward pass.

Since we use tri-linear interpolation to sample α, the surrounding eight grid cells of $\mathbf{x}_i$ should be updated with their corresponding tri-linear weights multiplied by the partial derivative $\partial L(\mathbf{x}, \boldsymbol{\omega})/\partial\alpha(\mathbf{x}_i)$.

4.3.3 Discussion on the constant environmental light. To ensure generality, we use a constant environmental light to render the medium. We do not choose point lights because they will generate uneven brightness and shadows that will be mistakenly regarded as useful features by the network. This negatively influences the albedo volumes predicted by the network. Fig. 6 shows the visual comparison

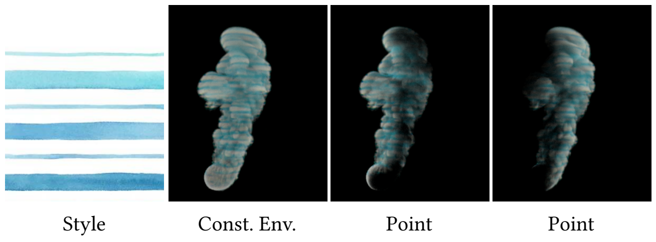

Fig. 6. The influence of lighting on the rendered image during training.

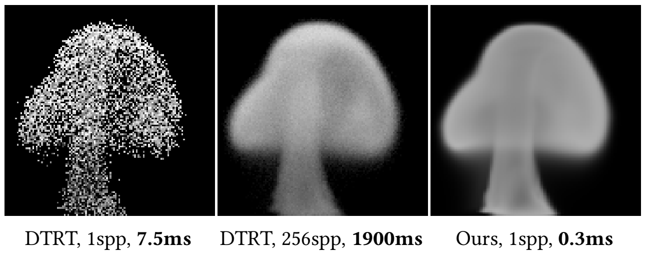

Fig. 7. Comparison between our volume rendering layer and DTRT [Zhang et al. 2019] in forward rendering. DTRT is too slow to be used in training deep neural networks since a high sampling rate is required.

of a stylized smoke plume lit by a constant environmental light (Const. Env.) and different point lights (Point), respectively. We can see that the synthesised image matches the corresponding style image closer under a constant environmental light. If we switch to point lights, color variations and shadows appear which differ greatly from the input style.

4.3.4 Comparison against the general method by Zhang et al. [2019]. Zhang et al. [2019] recently has proposed a general-purpose and physics-based differentiable volume rendering framework (DTRT) that can achieve the same goal as ours theoretically. Compared with our customized solution, DTRT is rather complicated and time-consuming both in the forward rendering and the backward gradient estimation. With the same scene configuration, our volume rendering layer is orders of magnitude faster than DTRT both at the forward stage and the backward stage, thanks to some precomputations and reasonable simplifications. Furthermore, their method requires a very high sample rate to achieve a noise level that is appropriate for network training, while our generated images are almost noise-free even at 1 sample per pixel (spp), as shown in Fig. 7. More importantly, the general method is impractically hard to converge because our application is a very high-dimensional optimization problem. Whereas, the linearity of the gradient in our volume rendering layer allows a much faster convergence rate.

4.4 Loss Function

To stylize an image, previous works usually employ the content loss and Gram loss. However, Risser et al. [2017] demonstrate the instability of Gram loss both experimentally and theoretically, and propose a more stable histogram loss for style transfer. In volumetric style transfer, we observe that the Gram loss converges slowly and easily causes artifacts. Considering this, we choose the histogram loss instead. To encourage spatial smoothness in the generated albedo volume, a 3D total variation (TV) loss is included [Johnson et al. 2016] in the loss function. Moreover, we also introduce a temporal loss to keep the stylized volumes coherent over time when a continuous sequence of volumes is being processed. Therefore, our overall loss function consists of three terms: a histogram loss, a TV loss and a temporal loss, defined as

$$\mathcal{L}(\mathbf{V}_\alpha, \mathbf{I}_s) = \lambda_{\text{hist}}\mathcal{L}_{\text{hist}}(\mathbf{V}_\alpha, \mathbf{I}_s) + \lambda_{\text{tv}}\mathcal{L}_{\text{tv}}(\mathbf{V}_\alpha) + \lambda_{\text{temp}}\mathcal{L}_{\text{temp}}(\mathbf{V}_\alpha, \mathbf{V}'_\alpha) \tag{12}$$

where λ_{hist}, λ_{tv} and λ_{temp} are used to balance the weight of each term. Currently, they are empirically set to 1, 10 and 300, respectively.

4.4.1 Histogram loss. The process of directly computing the histogram is not differentiable and is hard to integrate into a neural network. Therefore, we opt to evaluate the difference between the original feature map and the histogram-matched feature map [Risser et al. 2017]. In this way, the histogram loss for a 2D image is computed by

$$\mathcal{L}_{\text{hist-2D}}(\mathbf{I}, \mathbf{I}_s) = \sum_{l \in \{l_s\}} \sum_{c} \|\mathbf{M}^l \odot (\mathcal{F}^l_c(\mathbf{I}) - H(\mathbf{M}^l \odot \mathcal{F}^l_c(\mathbf{I}), \mathcal{F}^l_c(\mathbf{I}_s)))\|_2^2 \tag{13}$$

where $\mathbf{I}$ is a stylized image, $\mathcal{F}^l_c$ is the cth feature map channel at layer l of VGG-16, and H is a histogram matching function that returns the feature map suitably scaled. $\{l_s\}$ is the set of VGG-16 layers used to compute the histogram loss. To improve the fidelity of generated histograms, a mask $\mathbf{M}^l$ is employed to remove unnecessary background in each feature map of the stylized image $\mathbf{I}$, with $\odot$ representing element-wise multiplication. Note that the mask $\mathbf{M}^l$ is properly downsampled via max pooling to match the size of each layer.

With our volume rendering layer, we are able to compute the histogram loss for a 3D albedo volume in the 2D image space. Specifically, we first project 3D albedo volumes to several 2D images after setting the viewpoint v, accompanied with the corresponding masks used in Eq. (13) [1]. Then, the histogram loss for each albedo volume is defined as the summation of the 2D histogram loss evaluated by the projected images and the style image, i.e.,

$$\mathcal{L}_{\text{hist}}(\mathbf{V}_\alpha, \mathbf{I}_s) = \sum_{v \in \{v_p\}} \mathcal{L}_{\text{hist-2D}}(R(\mathbf{V}_\sigma, \mathbf{V}_\alpha, v), \mathbf{I}_s) \tag{14}$$

where v is a viewpoint selected from the viewpoint set $\{v_p\}$ and R denotes our volume rendering layer that generates a synthesized 2D image of the medium with $\mathbf{V}_\sigma$ and $\mathbf{V}_\alpha$ at the viewpoint v. Currently, we choose four views randomly sampled from a predefined camera path during each update. Since the synthesized images contain high dynamic range (HDR) information, they should be gamma corrected before feeding into VGG-16. To make training stable, we adopt the gamma correction method proposed by Kettunen et al. [2019] as a post-processing step of R.

[1]Each mask is generated by thresholding the corresponding rendered image with a constant 0.001.

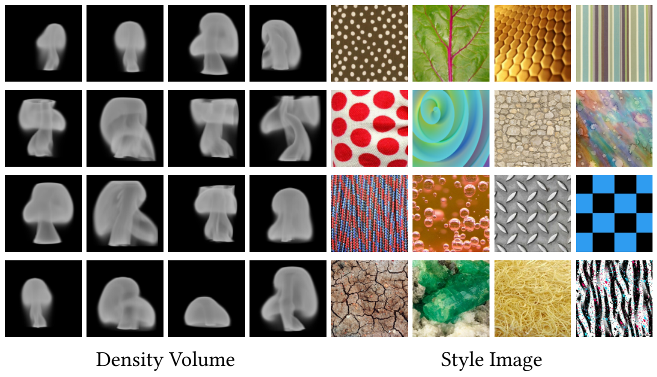

Fig. 8. Selected examples from our two datasets. The density volumes are visualized by our volume rendering layer.

Furthermore, we find that only measuring the histogram loss in the feature space will slightly lower the values of the stylized albedo. To address this problem, we add an additional histogram loss measured in the RGB space. This makes the brightness of rendered image closer to the input style image without loss of details.

4.4.2 Total variation loss. The TV loss is widely used in recent image style transfer methods [Johnson et al. 2016; Risser et al. 2017; Song et al. 2019], which serves as a regularization item, encouraging spatial smoothness in the generated image. We extend 2D TV loss into 3D as

$$\mathcal{L}_{\text{tv}}(\mathbf{V}_\alpha) = \|\nabla_x \mathbf{V}_\alpha\|_2^2 + \|\nabla_y \mathbf{V}_\alpha\|_2^2 + \|\nabla_z \mathbf{V}_\alpha\|_2^2 \tag{15}$$

where ∇_x, ∇_y and ∇_z are finite difference operators for the x, y and z direction, respectively.

4.4.3 Temporal loss. The temporal loss is designed to penalize the inconsistencies between stylized volumes of two consecutive frames. It is evaluated by

$$\mathcal{L}_{\text{temp}}(\mathbf{V}_\alpha, \mathbf{V}'_\alpha) = \|\mathbf{V}_\alpha - \mathcal{W}(\mathbf{V}'_\alpha, \mathbf{U})\|_2^2 \tag{16}$$

where $\mathbf{V}_\alpha$ and $\mathbf{V}'_\alpha$ represent two consecutive albedo volumes in an animation sequence. The warping function $\mathcal{W}$ uses the velocity field $\mathbf{U}$ to warp the previous albedo volume $\mathbf{V}'_\alpha$ to the current frame, eliminating the motion between the two frames. Their differences are measured by L_2 norm. Semi-Lagrangian advection is used to perform forward warping. Although a pair of temporally consecutive frames are required to compute the temporal loss during training, only one frame is needed as input in the prediction phase. Therefore, the temporal loss will not lower the performance of prediction.

4.5 Dataset

Our dataset consists of two parts: a density volume dataset and a style image dataset. The former contains 3000 density volume pairs (including the corresponding velocity fields) with a resolution of $64 \times 64 \times 64$ while the latter contains 3000 style images with a resolution of 128×128. Some examples from these two datasets are illustrated in Fig. 8.

We generate our density volume dataset with *mantaflow* [Thuerey and Pfaff 2018]. To ensure that the dataset covers a sufficient range of density patterns, we simulate 100 groups of smokes with a resolution of $64 \times 64 \times 64$. Each group contains 90 frames. We further simulate 20 groups of smokes with a higher resolution: $128 \times 128 \times 128$, to avoid the network overfitting to low-resolution volumes and make it more robust. The initial location, direction and speed for jetting smoke are disturbed randomly. For each group of low-resolution smokes, we take every two consecutive density volumes between the 61st and the 90th frames as a training example, together with the former's velocity field. For those high-resolution smokes, we perform a 2-step sampling. First, we also pair every two consecutive density volumes starting from the 61st frames. Then, we randomly crop 5 patches from each pair of density volumes. The style image dataset is selected from the DTD dataset [Cimpoi et al. 2014].

4.6 Training Details

Currently, our pipeline is implemented in PyTorch [Paszke et al. 2017] and trained jointly with the Adam optimizer [Kingma and Ba 2015] using a learning rate 0.0002 and a batch size 8. The hyper-parameters β_1 and β_2 of Adam are set to 0.9 and 0.999, respectively. We decay the learning rate by the power of 0.8 for every 50 epochs. The training process generally converges after 400 epochs, which takes about 76 hours on a NVIDIA Tesla V100 GPU.

The intermediate images produced by our volume rendering layer have a resolution of 160×160. Before fed into VGG-16, all images are normalized with mean $\{0.485, 0.456, 0.406\}$ and standard deviation $\{0.229, 0.224, 0.225\}$. We evaluate the histogram loss at layers *relu*1-2, *relu*2-2, *relu*3-3 and *relu*4-3 of a pre-trained VGG-16 network. The input style features (the means and variances of feature maps) of SKP are computed at the same layers.

5 RESULTS AND DISCUSSION

In this section, we conduct extensive experiments to validate our framework. Although our SKPN is trained with $64 \times 64 \times 64$ volumes, it can be applied to other resolutions as well. For the application of translucent material hallucination, a resolution of $128 \times 128 \times 128$ is used to voxelize the static models. For other smokes, we use a default resolution of $128 \times 256 \times 128$. Note that these density volumes and the style images presented in this section are not used in training. All images are synthesized by the Mitsuba renderer [Jakob 2010]. Please refer to the supplemental video for more results.

5.1 Comparison with Previous Methods

We first compare our SKPN to previous methods that support appearance stylization. In Fig. 9, we respectively compare our method to

- *PhotoWCT* [Li et al. 2018b]: a recent image style transfer method that is able to process arbitrary new style,
- *LazyFluid* [Jamriška et al. 2015]: a 2D patch-based approach to appearance transfer for fluid animations, and
- *Tomography* [Klehm et al. 2014]: a tomographic reconstruction method for volumetric appearance stylization.

PhotoWCT [Li et al. 2018b], as an image style transfer method, is only able to transfer color features to 2D images. Here, we supply PhotoWCT with the images rendered with density volumes and a

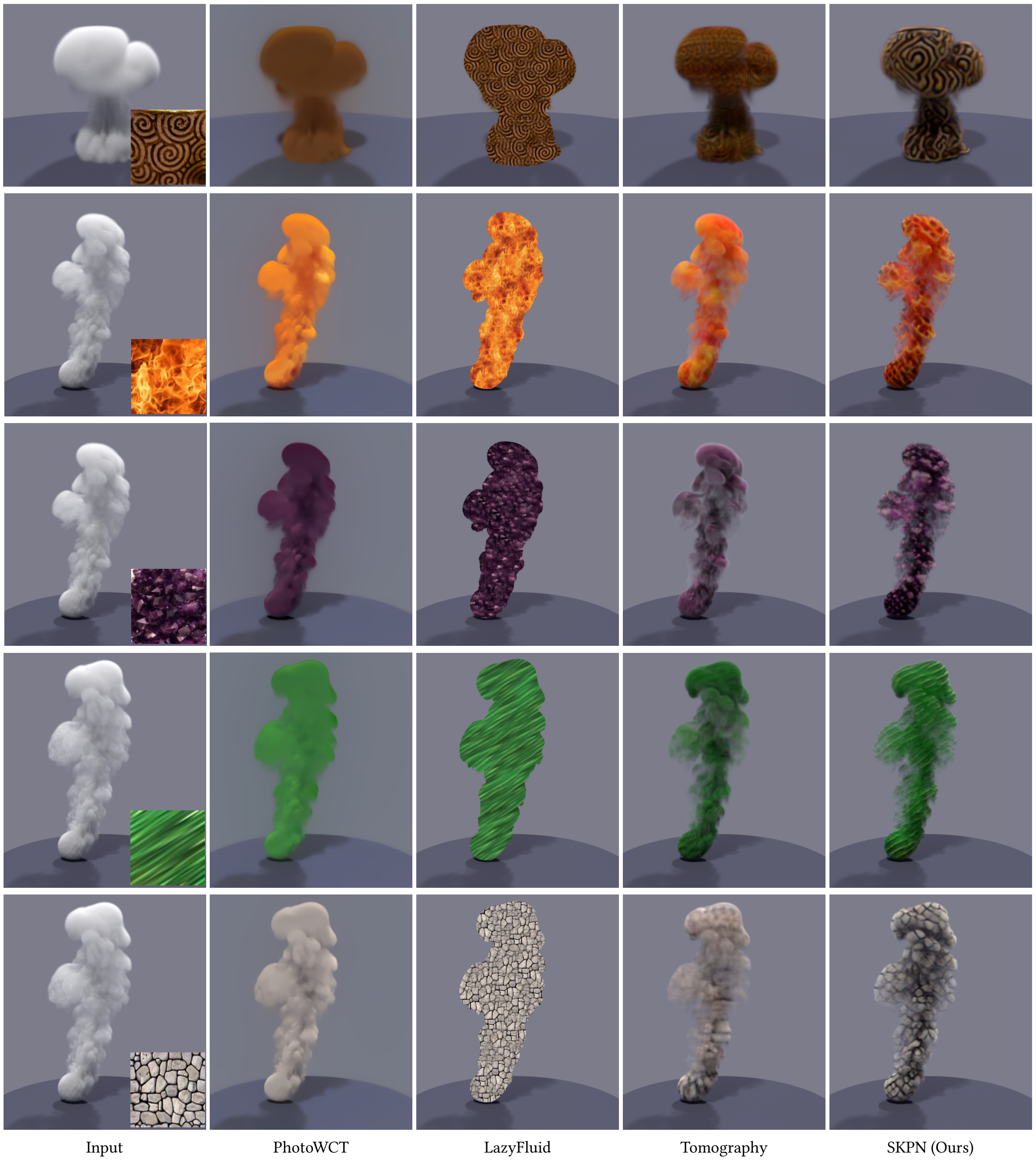

Fig. 9. Visual comparison with state-of-the-art methods on appearance stylization. The methods in comparison are PhotoWCT [Li et al. 2018b], LazyFluid [Jamriška et al. 2015], Tomography [Klehm et al. 2014], and the proposed SKPN.

Table 1. The percentage (%) of each method that gets the "best" vote in the user study.

Method	PhotoWCT	LazyFluid	Tomography	SKPN
Percentage	7.8	21.0	26.9	44.3

constant albedo 0.9. To guarantee the results where the semantically similar regions are stylized consistently, PhotoWCT includes a smoothing step that prevents it to capture sufficient details from our rendered smooth images. Although it has a certain degree of stereoscopic effects, the appearance is overly blurred. In the supplemental video, we also compared our SKPN with the work of Gatys et al. [2016] that can transfer great details. LazyFluid [Jamriška et al. 2015] is designed to transfer appearance for fluid animations based on flow-guided texture synthesis. This method is also restrict to 2D flows although it enables detailed feature control and improves temporal coherence. The tomography-based volumetric appearance stylization method [Klehm et al. 2014] relies on a linear optimization and multi-view images for high-quality volumetric reconstruction. Here, we feed this method with an identical image at different viewpoints. This leads to low-quality volumetric reconstruction due to the contradiction at multiple views. In comparison, our method based on SKPN achieves physically-plausible reconstruction even without correct multi-view images. Furthermore, since this tomography-based method is originally designed for static volumes, it will generate unnatural temporal variations for time-evolving fluids, as shown in the supplemental video.

For quantitative evaluation, we conduct a user study from 40 volunteers. The methods in comparison are shown in Fig. 9. We randomly choose 6 style images and 6 density volumes, which would obtain 36 results for each method. Each time, we display stylized results by these four compared methods side-by-side on a webpage in a random order, together with the corresponding style image and the visualization of the density volume. Participants are instructed to choose the "best" result for each style, based on the realism and style similarity. They are required to finish the task within 30 minutes, allowing zooming. The percentages of these methods in comparison are listed in Table 1. As seen, SKPN is mostly voted as the "best" one, demonstrating that our method significantly outperforms the others.

Fig. 10 compares our SKPN with LNST [Kim et al. 2020], the state-of-the-art neural style transfer algorithm that is tailored for 3D volumes. As an improved version of TNST [Kim et al. 2019], LNST is mainly designed to stylize structures in 3D fluids (see Fig. 10(c)). Due to its Lagrangian setting, it also supports color transfer from 2D images to fluid particles. However, it directly stores color information at each particle and thus neglects any translucent effect. Fig. 10(d) demonstrates transferring color information from a style image to 2D particles using LNST. Note that only 2D color transfer is shown in [Kim et al. 2020] and supported by their current implementation [2]. Concerning runtime performance, our SKPN is a feed-forward neural network trained end-to-end, enabling

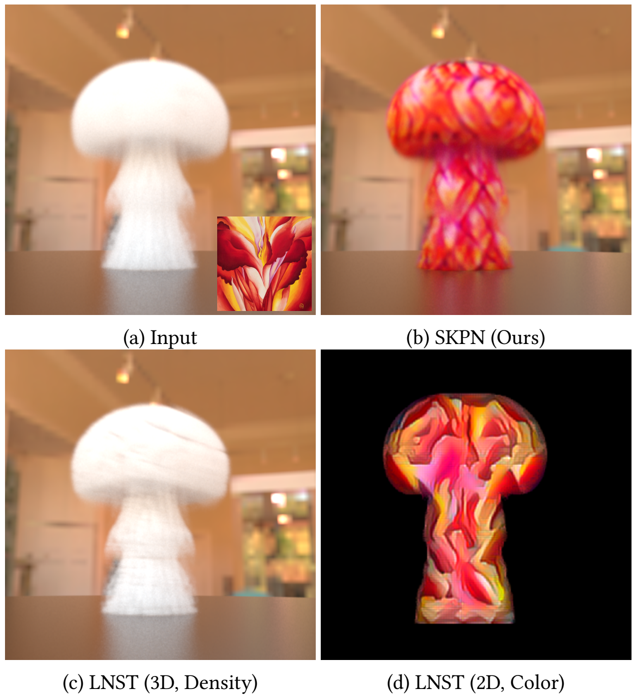

Fig. 10. Visual comparison between SKPN and LNST [Kim et al. 2020].

much higher performance compared to LNST which still takes a cumbersome non-linear optimization as an essential step.

5.2 Evaluation of the Loss Function

To demonstrate the effectiveness of our loss function, we conduct several ablation studies. One of the key ingredients in our loss design is the choice of the histogram loss instead of the Gram loss to guide the style transfer process. This makes the network training stable and tends to generate physically-plausible results, as validated in Fig. 11 and also explained by Risser et al. [2017]. If we replace the histogram loss with the Gram loss or combine them together, network training becomes instable and converges slowly, as show in the top row of Fig. 11. Here, each loss curves are normalized by dividing its first value. Visual comparisons in the bottom row also reveal that the Gram loss is not helpful to our volumetric style transfer problem. As seen, artifacts such as broken patterns easily occur if Gram loss is involved. In fact, using the histogram loss alone to encode style information is sufficient to preserve most details of the input style image. We also compare our histogram loss against traditional L_2 loss on mean and variance (i.e., mean-std loss). Although these two cases have the similar convergence rate, our model achieves more reasonable results in terms of visual appearance, because the histogram loss is more expressive than the mean-std loss. As shown in the bottom row of Fig. 11, the stylized appearance generated by the histogram loss has a closely matched color distribution with the input style image.

Another key term in our loss function is the temporal loss which is critical to enforce temporal coherence for time-evolving volumes.

[2]https://github.com/byungsook/neural-flow-style

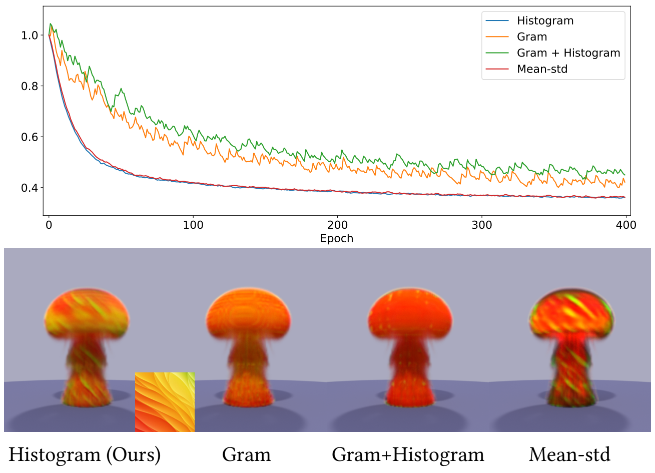

Fig. 11. Validation of the loss function. Top row: Evolution of test loss over the number of epochs for different loss functions. Bottom row: Impact of different loss functions on the transferred material appearance.

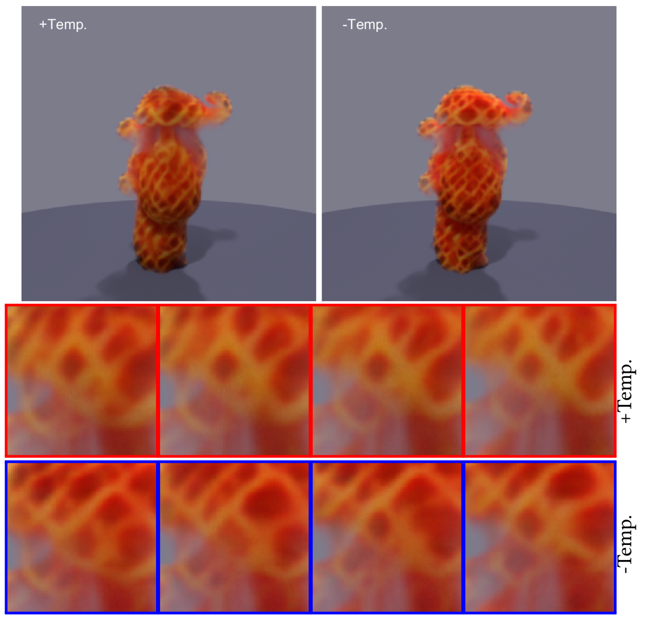

Fig. 12. Comparison between with (+Temp.) and without (-Temp.) temporal loss in training SKPN. Without temporal loss, flickering results are easily produced as highlighted in the last two rows. Please use Adobe Acrobat and click the first row to see the animation.

As shown in Fig. 12, the model trained without the temporal loss is prone to producing flickering results. The temporal flickering is eliminated by incorporating the temporal loss in training. In the closeups of Fig. 12, four consecutive frames in an animation sequence are shown and stylized with the same style image. In the supplemental video, we further show that SKPN trained with the temporal loss tends to produce consistent and stable stylized animations. A more efficient discriminative temporal loss has been recently proposed in tempoGAN [Xie et al. 2018]. However, ground-truth albedo volumes are necessary for training a discriminator, which are absent in our unsupervised settings. Nevertheless, the proposed temporal loss has already been able to guarantee temporal coherence.

5.3 Discussion on Kernels

In Fig. 13 we demonstrate the effectiveness of adopting multi-scale stylizing kernels in SKP. As aforementioned, using only one kernel (the leftmost one in Fig. 3) can not preserve the patterns well since it has less flexibility than multi-scale kernels. This is evidenced in the first column of Fig. 13 where we see that using one kernel fails to capture high-level structures from the input style images. As expected, double the kernels will make the encoded style more expressive and hence improve the quality of stylization. When four kernels are used, the appearance of the stylized volume is quite similar to the input style. However, adding more kernels will significantly increase the model size, making network training harder. For instance, adding one more kernel with $C_{\text{in}} = C_{\text{out}} = 128$ will incur over 4.6 million tunable parameters for our current network. Considering this, we opt to use four kernels which produce satisfactory results in most cases.

Currently, we choose $1 \times 1 \times 1$ kernels in each scale. This can further reduce the size of the trained model while still preserving the expressiveness of the stylizing kernels, as compared in Fig. 14. As seen, a model with $1 \times 1 \times 1$ kernels achieves comparable and even better results than that with $3 \times 3 \times 3$ kernels. Considering that the former possesses far less parameters than the latter (13 million vs. 134 million), it will reduce the risk of overfitting.

Our stylizing kernel can be viewed as an extension of AdaIN [Huang and Belongie 2017]. AdaIN assumes uncorrelation between feature channels and hence each channel is updated independently. Due to the additional convolutional layer in our stylizing kernel, we can inject style information across different feature channels, which modulates the feature more efficiently. This design is more suitable for stylizing 3D contents. As compared in Fig. 15, volumetric appearance stylization with specially-designed stylizing kernels outperforms AdaIN in preserving color patterns.

5.4 Impact of Views

The number of views used in network training also has a great impact on the final appearance of stylized volumes. Currently, we choose to train our SKPN with four different views when evaluating the histogram loss in Eq. (14). This helps to reproduce the desired appearance from any novel viewpoint, as shown in the first row of Fig. 16. If only one view is used during training, the network will generate stylized volumes with improper appearance. As seen, important features are absent at the viewpoints other than the original one.

5.5 Controlling the scale of structures

Our method has the ability to control the scale of stylized structures. One way to achieve this is by adjusting the resolution of density volumes, as shown in Fig. 17. To generate denser structural patterns

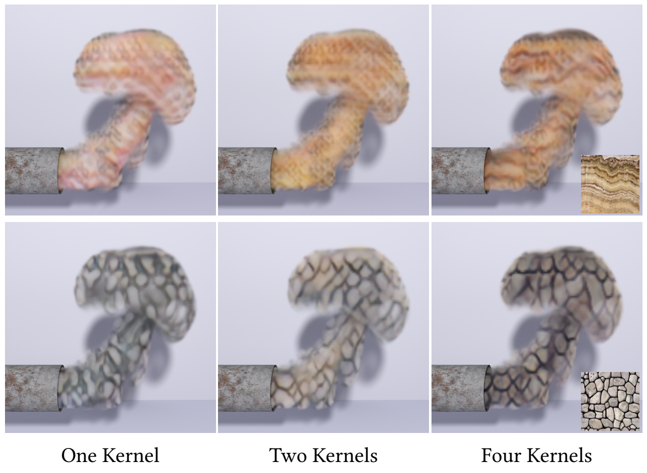

Fig. 13. Impact of the number of kernels on the visual effects. Note color drift and odd patterns produced by insufficient kernels (one or two).

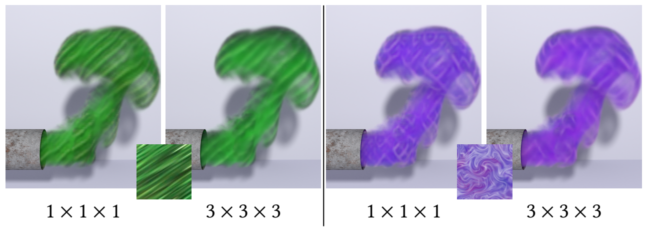

Fig. 14. Impact of the kernel size on the visual effects. Little difference is observed between kernel size of $1 \times 1 \times 1$ and $3 \times 3 \times 3$.

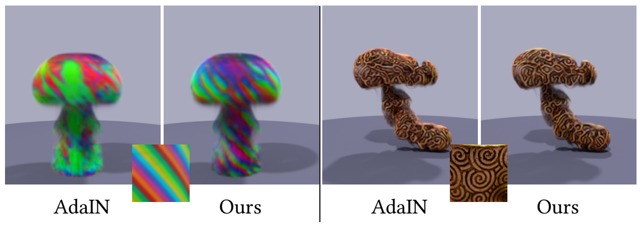

Fig. 15. Comparison between our stylizing kernel and AdaIN [Huang and Belongie 2017] in volumetric appearance stylization.

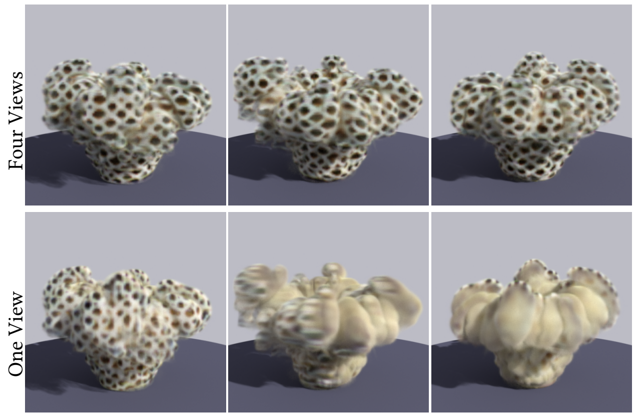

Fig. 16. Comparison between using four views (the first row) and one view (the second row) in evaluating the histogram loss during training. Training with four different views let the network faithfully reproduce the appearance from any novel viewpoint.

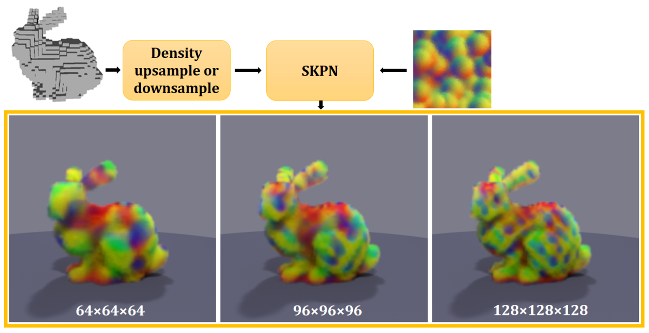

Fig. 17. Controlling the scale of structures by adjusting (upsampling or downsampling) the resolution of density volumes.

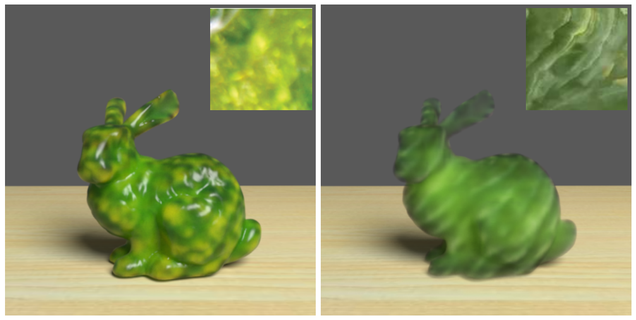

Fig. 18. Simulating the appearance of jade by hallucinating the translucent material from an input image taken from real world.

with fine details, we simply increase the resolution of the input density volume by bilinear upsampling. Then, the upsampled density volume is fed into SKPN, producing a stylized albedo volume with denser structures. Conversely, a downsampled density volume yields coarser structures than the original one.

5.6 Applications

Our SKPN proposes a new methodology that facilitates several graphical applications.

5.6.1 Hallucinating Translucent Materials. We first show its usage in translucent material hallucination. Hallucinating heterogeneous translucent materials (e.g., marble or jade) is a useful graphical application but is rather difficult [Schmidt et al. 2016; Song et al. 2009]. Conventionally, the materials need to be adjusted manually by rendering them repeatedly with different parameter settings. In Fig. 18, we demonstrate that the trained SKPN provides a goal-based

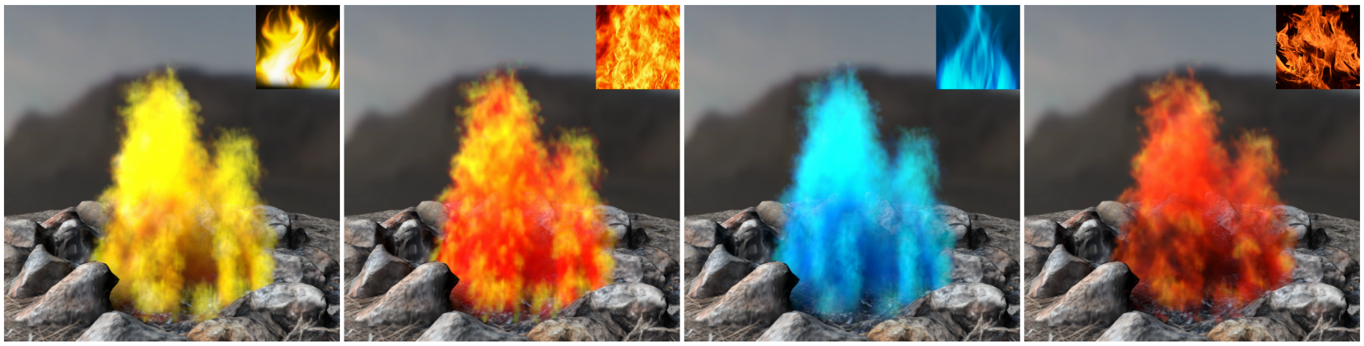
Fig. 19. Simulating dynamic fire flames with various styles (shown in the inset). Please use Adobe Acrobat to see the animation.

interface for efficient hallucination of translucent materials based on only one input image. Here, we try to automatically recreate a plausible translucent material from each image (shown in the inset) that would correspond to the image. To this end, we first voxelize the geometric model and feed it to our SKPN, together with a guided image serving as the style image. The generated albedo volume and the density volume (the voxelized model), possibly with a clear coating layer, encode a translucent material. The rendering results shown in Fig. 18 reveal that the hallucinated material closely matches the input image in each case. Owing to the high efficiency of SKPN, interactive hallucination is allowed. More hallucination results are shown in Fig. 1.

5.6.2 Simulating Fire Flames. Fires are widely used in visual effects and video games. Among many amorphous natural phenomena, they are considered to be the most difficult to describe because of their complex behaviors both in shape and appearance. Here, we show that our method facilitates the simulation of colorful fire flames with varying spectral properties probably caused by different chemical species. All we need are a dynamic flow representing the evolution of fire shapes and a reference image captured from real scenes or designed by artists. The dynamic flow can be produced by any existing fluid simulation tools like *mantaflow* [Thuerey and Pfaff 2018] and *Blender* [2020]. Without relying on any manually tuned parameter, even a novice user can easily create truth rendering of any type of fires with desired appearance, as demonstrated in Fig. 19. Here, the fire sequence is simulated by *Blender* with an output resolution of $128 \times 128 \times 128$. Note the matched fire appearance with the input style and the temporal coherence guaranteed by our method. Another scene is shown in Fig. 1. Please see the supplemental video for the full animations.

5.7 Runtime Performance

We evaluate the runtime performance on a PC with a 3.6 GHz Intel Core i7 processor and an NVIDIA GTX 2080Ti GPU. Since we put the computational burden at training stage, the stylization process at inference stage is very fast. In particular, only one density volume is required at inference stage, without its neighboring frames or velocity field. Our network takes about 0.04s, 0.25s and 0.51s on average to stylize a density volume with the resolution $64 \times 64 \times 64$, $128 \times 128 \times 128$ and $128 \times 256 \times 128$, respectively. As a comparison, the tomographic reconstruction method of Klehm et al. [2014] takes about 1.6s while LNST [Kim et al. 2020] takes several seconds (depending on the number of particles) on the same platform and using the same scene configuration.

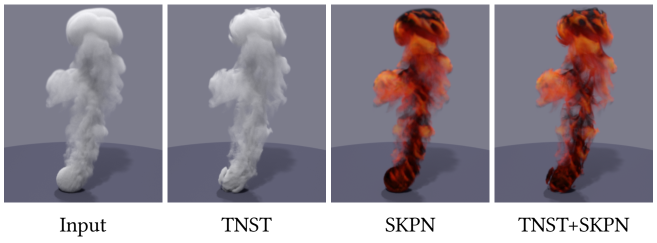

Fig. 20. Styling an input volume (using the fourth style image in Fig. 19) with TNST [Kim et al. 2019], SKPN and their combination, respectively. Note that color and structural features are not consistent when both SKPN and TNST are applied.

6 CONCLUSION AND FUTURE WORK

We have proposed *stylizing kernel prediction network* (SKPN), the first feed-forward neural network that enables efficient artistic volumetric appearance stylization. We formulate this as a volumetric style transfer problem in which a 3D density volume is stylized by an input style image. The key to our SKPN is the introduction of stylizing kernels and two carefully designed subnetworks, namely, *stylizing kernel predictor* (SKP) and *volume autoencoder* (VolAE). Once trained jointly, SKP produces a group of multi-scale stylizing kernels for a new style image, while VolAE accepts these kernels in its decoder and outputs a stylized albedo volume. Our results demonstrate that the stylized albedo volume achieves the similar appearance with the corresponding style image. The special design of the stylizing kernels and the loss function allow SKPN to handle arbitrary style transfer and generate temporally coherent animation sequences.

Our framework has several limitations that may inspire future research in this direction. First, since we use mean and variance statistics of feature maps to encode styles, the tiny details such as furs, fibers and grains in style images can not be well captured. It would be possible to include additional summary statistics (e.g., higher order moments) to enhance the details. Second, our SKPN is specifically designed to transfer color styles, keeping the input density volumes unchanged. To stylize both density and albedo of an input volume, one straightforward solution is to combine our proposed SKPN with TNST [Kim et al. 2019] (or LNST [Kim et al. 2020]), as illustrated in Fig. 20. However, it can not guarantee consistency between color and structural features if these two methods are applied independently. In the future, we would like to explore new methodologies to edit the density volumes with both color and structural features in a correlated way, enabling more powerful and harmonious style design.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. This work was supported by NSFC (62032011 and 61972194).

REFERENCES

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017)* 36, 4 (July 2017).

Sai Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph.* 39, 6 (2020), 245:1–245:18.

Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, , Goktekin, and James F. O'Brien. 2006. A Texture Synthesis Method for Liquid Animations. In *Proceedings of Eurographics/ ACM SIGGRAPH Symposium on Computer Animation.*

Blender. 2020. *Blender - a 3D modelling and rendering package.* Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

Robert Bridson. 2015. *Fluid Simulation for Computer Graphics.* CRC Press.

Robert Bridson, Ronald Fedkiw, and Matthias Müller-Fischer. 2006. Fluid Simulation. In *ACM SIGGRAPH 2006 Courses (SIGGRAPH '06).* Association for Computing Machinery, New York, NY, USA, 1–87.

Robert Carroll, Ravi Ramamoorthi, and Maneesh Agrawala. 2011. Illumination Decomposition for Material Recoloring with Consistent Interreflections. *ACM Trans. Graph.* 30, 4 (2011).

Eva Cerezo, Frederic Pérez, Xavier Pueyo, Francisco J. Seron, and François X. Sillion. 2005. A survey on participating media rendering techniques. *The Visual Computer* 21, 5 (Jun 2005), 303–328.

Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. 2017. StyleBank: An Explicit Representation for Neural Image Style Transfer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Tian Qi Chen and Mark Schmidt. 2016. Fast Patch-based Style Transfer of Arbitrary Style. *CoRR* abs/1612.04337 (2016).

Ming-Ming Cheng, Xiao-Chang Liu, Jie Wang, Shao-Ping Lu, Yu-Kun Lai, and Paul L. Rosin. 2020. Structure-Preserving Neural Style Transfer. *IEEE Transactions on Image Processing* 29 (2020), 909–920.

Fabienne Christen, Byungsoo Kim, Vinicius C. Azevedo, and Barbara Solenthaler. 2020. Neural Smoke Stylization with Color Transfer. (may 2020).

Mengyu Chu and Nils Thuerey. 2017. Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors. *ACM Trans. Graph.* 36, 4 (2017).

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. 2014. Describing Textures in the Wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).*

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09.*

Yoshinori Dobashi, Wataru Iwasaki, Ayumi Ono, Tsuyoshi Yamamoto, Yonghao Yue, and Tomoyuki Nishita. 2012. An Inverse Problem Approach for Automatically Adjusting the Parameters for Rendering Clouds Using Photographs. *ACM Trans. Graph.* 31, 6 (Nov. 2012), 145:1–145:10.

Bo Dong, Yue Dong, Xin Tong, and Pieter Peers. 2015. Measurement-Based Editing of Diffuse Albedo with Consistent Interreflections. *ACM Trans. Graph.* 34, 4 (July 2015). https://doi.org/10.1145/2766979

David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2002. *Texturing and Modeling: A Procedural Approach* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01).* Association for Computing Machinery, New York, NY, USA, 15–22.

Jonathan Gagnon, François Dagenais, and Eric Paquette. 2016. Dynamic lapped texture for fluid simulations. *The Visual Computer* 32 (05 2016). https://doi.org/10.1007/s00371-016-1262-8

L. A. Gatys, A. S. Ecker, and M. Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2414–2423.

S. W. Hasinoff and K. N. Kutulakos. 2007. Photo-Consistent Reconstruction of Semitransparent Scenes by Density-Sheet Decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 5 (2007), 870–885.

Milovš Hašan and Ravi Ramamoorthi. 2013. Interactive Albedo Editing in Path-Traced Volumetric Materials. *ACM Trans. Graph.* 32, 2 (April 2013).

K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 770–778.

L. G. Henyey and J. L. Greenstein. 1941. Diffuse radiation in the Galaxy. *The Astrophysical Journal* 93 (Jan 1941), 70–83.

G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507. https://doi.org/10.1126/science.1127647

Xun Huang and Serge Belongie. 2017. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, In International Conference on Computer Vision (ICCV), Venice, Italy (2017-10-22). *International Conference on Computer Vision (ICCV), Venice, Italy.* https://vision.cornell.edu/se3/wp-content/uploads/2017/08/adain.pdf Oral.

Ivo Ihrke and Marcus Magnor. 2004. Image-Based Tomographic Reconstruction of Flames. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '04).* Eurographics Association, 365–373.

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Ondřej Jamriška, Jakub Fišer, Paul Asente, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. 2015. LazyFluids: Appearance Transfer for Fluid Animations. *ACM Transactions on Graphics* 34, 4, Article 92 (2015).

Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. 2019. Neural Style Transfer: A Review. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 694–711.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Markus Kettunen, Erik Härkönen, and Jaakko Lehtinen. 2019. Deep Convolutional Reconstruction for Gradient-domain Rendering. *ACM Trans. Graph.* 38, 4 (July 2019), 126:1–126:12.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2019. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 188.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2020. Lagrangian Neural Style Transfer for Fluids. *ACM Transactions on Graphics* 39, 4, Article 52 (2020), 10 pages. https://doi.org/10.1145/3386569.3392473

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

O. Klehm, I. Ihrke, H. Seidel, and E. Eisemann. 2014. Property and Lighting Manipulations for Static Volume Stylization Using a Painting Metaphor. *IEEE Transactions on Visualization and Computer Graphics* 20, 7 (2014), 983–995.

V. Kwatra, D. Adalsteinsson, T. Kim, N. Kwatra, M. Carlson, and M. Lin. 2007. Texturing Fluids. *IEEE Transactions on Visualization and Computer Graphics* 13, 5 (2007), 939–952.

Chuan Li and Michael Wand. 2016. Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 702–716.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018a. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37, 6 (2018), 222:1–222:11.

Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. 2017a. Universal Style Transfer via Feature Transforms. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA.* 386–396.

Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. 2018b. A Closed-Form Solution to Photorealistic Image Stylization. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 468–483.

Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. 2017b. Demystifying Neural Style Transfer. arXiv:1701.01036 [cs.CV]

Hsueh-Ti Derek Liu, Michael Tao, and Alec Jacobson. 2018. Paparazzi: Surface Editing by way of Multi-View Image Processing. *ACM Transactions on Graphics* (2018).

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning. In *The IEEE International Conference on*

Computer Vision (ICCV).

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 154–169.

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing Discontinuous Integrands for Differentiable Rendering. *ACM Trans. Graph.* 38, 6, Article 228 (Nov. 2019), 14 pages.

Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. 2017. Deep Photo Style Transfer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. 2018. Burst Denoising With Kernel Prediction Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Trans. Graph.* 38, 6 (2019).

Makoto Okabe, Yoshinori Dobashi, Ken Anjyo, and Rikio Onai. 2015. Fluid Volume Modeling from Sparse Multi-view Images by Appearance Transfer. *ACM Transactions on Graphics (Proc. SIGGRAPH 2015)* 34, 4 (2015), 93:1–93:10.

Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. 2018. PhotoShape: Photorealistic Materials for Large-Scale Shape Collections. *ACM Trans. Graph.* 37, 6, Article 192 (Nov. 2018).

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch.

Felix Petersen, Amit H. Bermano, Oliver Deussen, and Daniel Cohen-Or. 2019. Pix2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer. arXiv:1903.11149 [cs.CV]

Eric Risser, Pierre Wilmot, and Connelly Barnes. 2017. Stable and Controllable Neural Texture Synthesis and Style Transfer Using Histogram Losses. arXiv:1701.08893 [cs.GR]

M. Ruder, A. Dosovitskiy, and T. Brox. 2018. Artistic style transfer for videos and spherical images. *International Journal of Computer Vision* 126, 11 (Nov 2018), 1199–1219. http://lmb.informatik.uni-freiburg.de/Publications/2018/RDB18 online first.

Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. 2018. Example-based Turbulence Style Transfer. *ACM Trans. Graph.* 37, 4 (July 2018), 84:1–84:9.

Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. 2016. State of the Art in Artistic Editing of Appearance, Lighting and Material. *Computer Graphics Forum* 35, 1 (2016), 216–233.

Falong Shen, Shuicheng Yan, and Gang Zeng. 2018. Neural Style Transfer via Meta Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

L. Shen, D. Zhu, S. Nadeem, Z. Wang, and A. E. Kaufman. 2018. Radiative Transport Based Flame Volume Reconstruction from Videos. *IEEE Transactions on Visualization and Computer Graphics* 24, 7 (2018), 2209–2222.

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015*.

Chunjin Song, Zhijie Wu, Yang Zhou, Minglun Gong, and Hui Huang. 2019. ETNet: Error Transition Network for Arbitrary Style Transfer. *Conference on Neural Information Processing Systems (Proceedings of NeurIPS 2019)* (2019).

Ying Song, Xin Tong, Fabio Pellacini, and Pieter Peers. 2009. SubEdit: A Representation for Editing Measured Heterogeneous Subsurface Scattering. *ACM Trans. Graph.* 28 (08 2009). https://doi.org/10.1145/1576246.1531337

Nils Thuerey and Tobias Pfaff. 2018. MantaFlow. *http://mantaflow.com*.

Adrien Treuille, Antoine McNamara, Zoran Popoviundefined, and Jos Stam. 2003. Keyframe Control of Smoke Simulations. *ACM Trans. Graph.* 22, 3 (2003), 716–723.

Dmitry Ulyanov, Vadim Lebedev, Andrea, and Victor Lempitsky. 2016a. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1349–1357.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. 2016b. Instance Normalization: The Missing Ingredient for Fast Stylization. *CoRR* abs/1607.08022 (2016).

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Trans. Graph.* 37, 4 (July 2018), 124:1–124:15.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow. *ACM Trans. Graph.* 37, 4 (2018).

Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-Space Differentiable Rendering. *ACM Trans. Graph.* 39, 4 (2020), 143:1–143:19.

Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A Differential Theory of Radiative Transfer. *ACM Trans. Graph.* 38, 6 (2019).