2019 IEEE Pacific Visualization Symposium (PacificVis)

DNN-VolVis: Interactive Volume Visualization Supported by Deep Neural Network

Fan Hong[1] * Can Liu[1] † Xiaoru Yuan[1,2] ‡

1) Key Laboratory of Machine Perception (Ministry of Education), and School of EECS, Peking University, Beijing, China
2) Beijing Engineering Technology Research Center of Virtual Simulation and Visualization, Peking University, Beijing, China

Abstract

In this work, we propose a novel approach of volume visualization without explicit traditional rendering pipeline. In our proposed method, volumetric images can be interactively 'reversed' given the volumetric data and a static volume rendered image under the desired rendering effect. Our pipeline enables 3D-navigation on it for exploring the given volumetric data without explicit transfer function. In our approach, deep neural networks, combined usage of Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNN) are employed to synthesize high-resolution and perceptually authentic images directly, inheriting the desired transfer function and viewing parameter implicitly given by the input images respectively.

Keywords: Deep learning, volume rendering, transfer function, generative adversarial network, machine learning

1 Introduction

Volumetric data ranging from medical diagnoses to various scientific simulation has been widely studied during the past 30 years. Direct volume rendering (DVR) is one of the most widely applied and practical techniques to explore volumetric data. One essential step in volume rendering is designing proper transfer functions, which map raw voxels values, scalar or multivariate, to color-opacity values [27]. Essentially, the transfer function acts as a classification function to identify volume features and give each feature specific color-opacity values. Appropriate transfer functions are critical for users to analyze and comprehend the intricate structures in the volumetric datasets.

In traditional volume rendering pipeline, the transfer function acts as a classification function to identify volume features with color and transparency values. When the viewpoint parameters are chosen, the color of each pixel in the images composites from the color-opacity values of the voxels.

However, rather than designing transfer functions start from scratch, sometimes we want to explore the volume in such cases: (1) In the literature about volumetric data, the rendering results are given, and the volumetric data is usually open-sourced, while the transfer function is often not published. The exploration under this rendering effect for the volumetric data is wanted. (2) During the exploration of volumetric data, after users designed transfer functions and got the rendering results, they may want to apply some modifications to the results for better presenting their goal. Modifying rendering results is more straightforward than adjusting transfer functions through a trial-and-error way, which is tedious and inefficient. The exploration under the modified rendering effect is desired.

In these scenarios, ideally, the transfer function underlying is desired to be recovered given a rendering result. However, unlike the pipeline from the transfer functions to results, the workflow reversed is an ill-posed problem. On the one hand, the transfer function design space is so tremendous that traversing that is impossible, then brute-force method to reverse the transfer functions is unfeasible. On the other hand, linking features in the rendered images to the corresponding voxels' density range and physical position is not a trivial task. It is not rare that those small modifications in the transfer function setting can be accumulated through a large number of composited voxels and result in significant changes in rendering results. Such nonlinear mapping makes it extremely difficult to reversely getting the transfer function.

To tackle these challenges in traditional transfer function design methods, some approaches proposed interactive ways to specify transfer functions from the results. Among early attempts to relieve the design burden from the user, Design Galleries [26] was proposed to let the user select desirable transfer functions from a series of pre-generated set of volume rendering images. However, Design Galleries method follows the traditional rendering pipeline to compute the pairs of transfer functions and corresponding effects. As the transfer function design space is vast, it is not possible to exhaust all possible ones. Iterative selections are often required until satisfactory results obtained. Other attempts, such as WYSIWYG volume visualization [9, 11], support direct interaction and editing over the volume rendered images. The WYSIWYG approach requires pre-computation of the topological structures of the volume data. Since it based on the users' limited actions to guess the desired effects, it's an attempt of half 'reverse' of the pipeline.

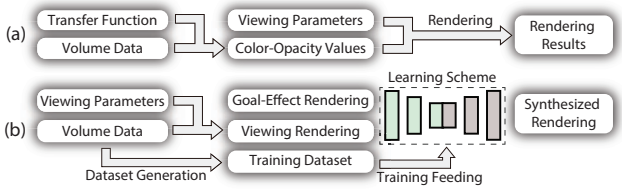

Figure 1: The comparision of traditional volume rendering pipeline and our proposed pipeline. (a) the pipeline of conventional volume rendering, (b) the pipeline of our approach.

In this work, we propose to take a dramatically different approach, employing deep neural network-based models to enable the interactive volume visualization more directly and effectively. As Figure 1 shows, unlike the traditional rendering pipeline, we treat the kernel of our interactive volume visualization task as a function which accepts the original images with goal rendering effects and new viewing parameters as the inputs and synthesizes new images. The new image is expected to emulate the rendering result of the specific volumetric data with the rendering effects of goal rendering under the new viewing parameters. We first generate a training dataset consists of rendering results based on random selection of transfer functions without any prior knowledge. The training set is then fed into a machine learning scheme, and an implicit model can be trained embedded with the necessary volume features. Our whole

*e-mail: fan.hong@pku.edu.cn
†e-mail: can.liu@pku.edu.cn
‡e-mail: xiaoru.yuan@pku.edu.cn (corresponding author)

2165-8773/19/$31.00 ©2019 IEEE
DOI 10.1109/PacificVis.2019.00041

pipeline is expected to approximate the function G:

$$I_{new} = G(I_{goal},\ V,\ D)$$

In which I_{new}, I_{goal}, V and D stand for the new image, the goal image, the viewing parameters, and the volumetric data respectively. The approximation capability is supported by existing learning theory, as neural networks with sufficient depth and width can approximate any differentiable functions with adequate training [14]. In this architecture, convolutional neural networks (CNNs) [23] and generative adversarial networks (GANs) [8] are coupled to achieve generation of images for volumetric data.

With a rendering provided by the user without explicitly knowing the features and color-opacity settings, our model synthesizes images inheriting the desired properties indicated in the rendering. As the user can specify viewing parameters, 3D navigation is enabled with our model to explore the corresponding volume data. Our proposed deep learning model acts as an implicit 'reversed' rendering pipeline for volume visualization by synthesizing images. We can use the goal-effect image to get the renderings under other viewing parameters without knowing the complex transfer functions.

We further build an interactive system for practical usage. Users can upload a rendering to express the desired rendering effects for a given specific volume data. The system can synthesize images inheriting the effects with 3D navigation activated accordingly. The uploaded image can be interactively explorable in our system, as the system can generate results from different viewing directions. Besides, users are allowed to apply modifications the uploaded image directly, e.g., adjusting color curves, contrast, and brightness. In such situations, without an explicit definition of the corresponding transfer functions, our approach achieves a high quality of the volume visualization results similar to traditional direct volume rendering.

2 Related Work

Our approach intends to replace the traditional transfer function design step in direct volume rendering. In this session, we will go through major efforts in related transfer function research. We also present backgrounds about deep learning techniques, especially image generation-related ones, which makes our new volume visualization possible.

2.1 Transfer Function Design

In the past decades, lots of techniques have been proposed to improve transfer functions for better visualization quality. Major transfer function designing approaches can be classified into two types, which are data-driven and result-guided respectively.

In data-driven approaches, transfer function design is considered as a classification problem, where voxels are first classified into several groups according to their value distribution, then are assigned colors and opacities based on their belonging classes. Early transfer functions are merely one-dimensional, which classify voxels only based on their scalar intensity values [13]. However, 1D transfer functions are prone to noise and challenging to provide satisfying results when dealing with complex volume data. Kindlmann and Durkin [19] suggested adding first- and second-order derivatives to construct 2D histograms for classification. Later, various features of voxels are derived for classification, such as gradient, second derivatives [21], spatial distances to boundaries [33], scale feature [3], occlusion [4], visibility [5], etc. Topology analysis [6] has also been considered to extract essential features for classification. As for multivariate volume data, they naturally involve more dimensions of features. High dimensional data visualization techniques, like dimensional reduction, parallel coordinates and scatter plots, have been employed for transfer function designing [1, 10]. With more derived properties of voxels involving in the classification, the features in the volume data are usually extracted and visualized more precisely. However, the user interaction in such advanced transfer function design also becomes more complicated and non-intuitive, which is challenging for novice users.

Most above mentioned transfer function designing approaches are done through trial-and-error interactions, which makes the interaction very time-consuming. There are several works which introduced advanced statistics or learning techniques for the classification problem to reduce users' burden. Independent component analysis [37] and principal component analysis [31] are employed to transform the original data to component feature space for classification and semantic structures respectively. In this way, users' efforts to define novel data features are reduced, more complicated feature vectors are derived for improving the classification accuracy, e.g., local frequency distribution [17], sparse coding from dictionary learning [28]. Learning-based models, like neural networks, are proposed by Tzeng et al. [38, 39] to assist the classification task with example voxels brushed by users. However, indirect correspondence between color and opacity settings and the rendering results still exist, as users still manipulate in the data space. In our approach, we employ advanced deep learning techniques, for direct image synthesis rather than for the classification task, leading to more direct corresponding to the image space.

As data-driven transfer function design approaches are suffered from the gap between the data space and the resulting images, result-guided approaches are more attractive as alternatives for effective volume visualization. Design galleries [26] generate rendering images sampling many transfer functions and embed them into a 2D space to let users select most satisfying ones. However, the efficiency and quality are strongly constrained as only limited samples can be extracted in the large possible transfer function design space. Guo et al. proposed a WYSIWYG way to directly manipulate in the image space so that the underlying features and corresponding transfer function are updated according to users' intention [9, 11]. In such approaches, users' manipulation is changed from the data space to the image space, but the actual computation space is still on the data side. Example-based volume illustration [25] synthesizes 3D textures by emulating styles of scientific illustrations. However, in that work, the extract color mapping from the given images cannot be fully restored, and only rough color styles are adapted. Wu and Qu [40] proposed an interactive transfer function design by editing on the rendering images. They assume 1D or 2D transfer functions are used and need to derive them from the images explicitly. In contrast, our approach does not have such presumption, which broadens its usage. Even for images edited with image-level operators, where transfer functions are not explicitly defined, our goal-oriented approach can still work well.

2.2 Image Generation

Our proposed approach relies on deep learning to generate an implicit model from the volume. As a part of machine learning methods, deep learning aims at learning data representations with deep neural networks. Deep learning architectures have been applied to various fields, and have achieved results which are comparable or even superior to human experts in some applications. Related to techniques of image generation with deep learning, two major network structures are frequently used: Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs).

Convolutional neural networks (CNNs) [23] is a type of deep neural networks which has been successfully applied to analyzing images. Ordinary feed-forward neural networks [34] are made up of multiple layers, each of which contains neurons with learnable weights and biases. However layers of CNNs are called convolutional layers, neurons are arranged in a 2D space, and the inputs of them are only a local region of the previous layer's output. The sparsity and shared weights significantly reduce the model size and present a better performance for image-specific tasks [12, 22, 36],

like image classification, object detection, etc. The whole network still approximates a single differentiable function: from the raw image pixels to certain outputs. Other layers like batch normalization layers [15], pooling layers are used in CNNs to improve the performance. Among state-of-the-art network architectures, ResNet [12] has introduced special skip connections that connect non-neighboring layers, leading to extraordinary performance, which inspires our design of network architecture.

Generative adversarial networks (GANs) is another type of architectures used in deep learning for unsupervised learning and first introduced by Ian Goodfellow et al. in 2014 [8]. The original GANs consists of two neural networks, a generator, and a discriminator, which compete with each other in a game framework. The generator tries to generate data which are similar to the latent distribution of training samples, while the discriminator attempts to discriminate whether the input is synthesized by the generator or is from the training set.

GANs are more training schemes than fixed architectures, where CNNs can be embedded for image generations. For the generator, convolutional layers can be used to generate seed vectors from images, while deconvolutional layers are used as an up-sampling technique to synthesize images from vectors [29], while for the discriminator, convolutional layers are employed to extract features from images for the discrimination task. With adequate training, GAN can generate images that are more photorealistic and authentic to humans' perception than previous techniques. Combined usage of GANs and CNNs already have lots of applications in image generation, such as image super-resolution [24] and domain transfer [16, 43]. GANs are even able to synthesize images conditioned by non-image data, such as texts [30, 42] and visual attributes [41], to fulfill specific goals. Such capability is especially suitable for our interactive volume visualization, which involves volume rendering conditions into our model.

3 Method Overview

In the following, we give a formal description of our goal-oriented volume visualization task as a differentiable function, and the special adaptation of viewpoints we made to improve the performance. We then introduce the overall Generative Adversarial Network (GAN)-based architecture of our deep learning model which is used to approximate the proposed function. As for the details, we give more information about (de)convolutional layers and their related techniques, and how they build the whole network. CNNs [23] are used to extract features of renderings (convolutional layers) and to synthesize new images (deconvolutional layers). At last, we describe how we train and validate our model with generated samples.

3.1 Problem Definition

In this work, we consider goal-oriented volume visualization with images indicating the desired transfer function effects. Our model, after adequate training, is expected to synthesize images with consistent styles as the provided ones under different view directions.

To formally define the above problem, we first review the traditional transfer function design process. It can be considered as a process of two steps. The first step is a classification problem of voxels. For every voxel indexed by its location (x,y,z), a feature vector $\vec{v}(x,y,z)$ is derived based. The feature vectors can be varied ranging from the voxel's properties like intensity value, its neighboring properties like gradients or local distributions, to global properties such as visibility or feature sizes. Then, the voxels are classified based on their feature vectors $\vec{v}(\cdot)$ through certain criteria.

In the second step, different categories are assigned colors and opacities. Each voxel obtains its color and opacity value based on its belonging categories and/or corresponding probabilities. Due to the variety of the classification results, we do not assume a simple representation of the color mapping. We represent the colored volume after the mapping as $C(x,y,z)$. At last, through certain rendering techniques in the direct volume rendering, an image is generated from the colored volume C.

In the whole process of the direct volume rendering, the transfer function designing actually defines two mappings starting from the original voxels, i.e. $(x,y,z) \rightarrow \vec{v}(x,y,z) \rightarrow C(x,y,z)$. However, as for our task of goal-oriented volume visualization, it is of great difficulty to reason the two mappings straightforwardly without prior knowledge of themselves. The feature vectors $\vec{v}(\cdot)$ as well as the classification criteria are hard to guess and extract from the images. Meanwhile, the composition and blending operation in the rendering pipeline makes it difficult to restore the original color mapping. The reduction of dimensions from 3D to 2D also causes significant spatial information loss. The processes of the transfer function design and direct volume rendering are severely lossy and hardly reversible, which makes it difficult to achieve goal-oriented volume visualization directly. Nonetheless, in this work, we propose a dramatically different approach by employing advanced deep learning techniques to tackle this challenge indirectly.

Our learning-based solution couples two sub-tasks: the estimation and approximation of the transfer function and the re-application under new viewpoints, i.e., (1) Image $I_1 \rightarrow$ Data D, Transfer function C, and Viewpoint V_1, (2) Data D, Transfer function C, and Viewpoint $V_2 \rightarrow$ Image I_2. Lighting effects and other post-processing effects are not considered for now. In our approach, we choose not to directly extract the transfer function C from the images I_1. Instead, we tackle the problem by integrating the two subtasks, that is to synthesize rendering images inheriting the transfer functions implicitly given by the images under different viewpoints. In this way, our trained model acts as an implicit rendering pipeline for volume visualization.

We represent the whole task as a function $G : (I_1, D, V_2) \rightarrow I_2$ by collecting and canceling the inputs and outputs in two sub-tasks. Since the data D is invariant in this task, we can further simplify the function as $G : (I_1, V_2) \rightarrow I_2$. Following the above task function $G : (I_1, V_2) \rightarrow I_2$, we have tried constructing a GAN-based network to approximate it. However, current state-of-the-art GANs are not able to generate satisfactory images with high resolutions, like 1024^2, without additional auxiliary information. Most of the existing works can only generate images with sizes up to 256^2 if quality not sacrificed. We decided to involve more information in the model inputs. To be specific, we choose a trivial transfer function, denoted as $C_0(\cdot)$, and render an image I_3 with C_0 under the expected viewpoint V_2 through traditional rendering pipeline. We name this step as pre-rendering. I_3 replaces V_2 as an input of the function, so the function to be estimated becomes $G : (I_1, I_3) \rightarrow I_2$. In the default transfer function C_0, the opacity function is tuned to show sufficient details of the volumetric data. The color function can be more flexible which only need to make those features distinguishable. The pre-rendering image I_3 mainly acts as supplementary information to original images I_1. Although I_3 may not cover every detail of the data, it together with I_1 improves the quality of synthesized images as our results demonstrate. In Section 5, we show our choices of default transfer functions for different datasets.

3.2 Overall Model Architecture

We will discuss the deep learning network architecture in this section as shown in Figure 2.

GANs can be considered as a training scheme rather than a fixed network, and they can be coupled with many other learning architectures. Convolutional layers are commonly used to extract features of images in the discriminator to help its discrimination task, and deconvolutional layers are used for up-sampling to synthesize images in the generator. In this part, we mainly discuss the adaptation we made on the overall model architecture comparing with the common version of GAN:

1. The synthesized images are generated from the *encoding vec-*

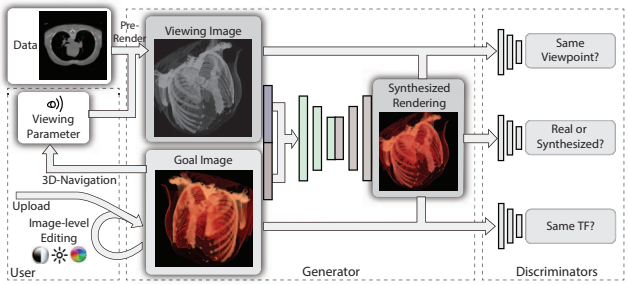

Figure 2: Workflow of our approach including the overall network architecture of our model.

tors of the inputs, instead of from random noise vectors. The encoding vectors are small vectors which convey important information extracted from the inputs. As suggested in many recent GAN-based models [24, 41], the inclusion of encoding vectors can provide direct controls of the generated images to make the images dependent on the inputs.

2. Following the previous one, our model accepts two image inputs at the same time, i.e., the goal image I_1 and the image with the setting viewpoint I_3. This makes the generated images be conditioned by multiple factors simultaneously, i.e., satisfying the function $G : (I_1, I_3) \rightarrow I_2$, to achieve our task.

3. Multiple discriminators are employed in our network to achieve our goal in multiple aspects. The original GAN and many derivatives have only one discriminator to compute whether the image is synthesized or true. However, very different from other previous work in image synthesis with the deep neural networks, the requirements for the synthesized images is defined with two folds of information, i.e., the desirable view parameters and transfer function matched with the example image. It is difficult to make simple loss functions to directly judge if two volume visualization result images under different viewing setting are consistent regarding the transfer function setting. To tackle such a challenge, we employ additional discriminators in the neural network structure. Specifically, in our scheme, one discriminator D_{TF} is defined to judge if the synthesized rendering is consistent with the goal image I_1 in terms of rendering effects. The other one D_{V} judges if the synthesized rendering has the same viewing setting as the input image I_3. As their discrimination ability improves, the generator also synthesizes images that are more fit to our goals.

4. The resolution of synthesized images is much larger than existing GAN-based models, which significantly increases the difficulty of model training. In our scheme, replacing input V_2 with a pre-rendering image I_3 assists a lot to the goal. Furthermore, to strengthen the reconstruction ability of our model, advanced techniques like skip connection-based models are adopted.

3.3 Model Details

In the section, we give more details about the network construction of the generator and discriminators. In the following description, *images* and *tensors* are interchangeable to represent 3-dimensional data with width, height, and channel.

Cross-Correlation Operator and (De)Convolutional Layers

Convolutional layers are widely used in image-related deep network architectures. The underlying mathematical operator is named as cross-correlation $\star$. Besides, settings such as stride, padding, dilation, etc., could be involved. Due to page limitation, we suggest referring to other literature for further information [7].

Convolutional layers are typically designed for spatially-organized data, such as images. In those layers, cross-correlation operators are applied on multi-channel 2D data (images or tensors) with a small-size kernel and obtain new multi-channel images. Following the settings in cross-correlation, a convolutional layer accepts inputs of size (C_{in}, H, W), where C_{in}, H, and W are the number of channels, height, and width. The output has a size of $(C_{out}, H_{out}, W_{out})$, which can be calculated from the following formula:

$$out(C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(k),$$

where $weight(\cdot)$ is the convolution kernels which are usually very small. $bias(\cdot)$ and $weight(\cdot)$ are the parameters in the convolutional layer. The computation property of convolutional layers makes it more suitable to extract local spatial features for the following tasks.

In our network, we choose the kernel size to be 4×4 and set the stride and padding to 2 and 1 respectively. With such setting, the width and height of images will be reduced to half after each layer. With multiple convolutional layers stacked, the image sizes become smaller and the abstraction levels become higher.

In contrast to convolutional layers, which can be considered as a down-sampling technique, deconvolutional layers, or formally called transposed convolutional layers, are employed for up-sampling. Briefly speaking, if with the same settings of size, stride and padding as the convolutional layers, the output image sizes of the transposed convolutional layers will double each time with data values are composited from its input. So with deconvolutional layers, it is possible to synthesize large images iteratively from small tensors.

Layer Blocks

In the network architecture, (de)convolutional layers are usually used with other layers forming a function block. As shown in Figure 3 and Figure 4, (de)convolutional layers are represented as red and blue squares above the gray arrows, respectively. The commonly-used blocks are (de)convolutional layer, BatchNorm (batch normalization) layer, and ReLU (rectified linear units) activation, represented with squares in other colors.

When training a model, multiple images are usually fed to the model to be processed simultaneously, which is called a batch. Batch normalization [15] layers usually follow (de)convolutional layers to normalize their outputs. Normalization operations could make the features in the data more outstanding. ReLU activation is defined as $f(x) = max(0, x)$, which has efficient gradient propagation and computation. Leaky ReLU is a variation of ReLU, defined as $f(x) = max(cx, x), c \in (0, 1)$, inheriting its advantages and can avoid never-activated dead neurons. No vanishing or exploding gradient problems exist for ReLU activation, compared to other activation functions like sigmoid. In our network architecture, we do not use pooling layers, although they are commonly used in many image-based models. This is because pooling layers result in a loss of information, which could be harmful to our image synthesis process.

Architectures of the Generator and Discriminators

The architecture of the generator, as shown in Figure 3 can be divided into two parts, encoding and decoding respectively. In the encoding process, the inputs are encoded into a small tensor through several layer blocks. In our work, the inputs are two images, i.e., the goal image I_1 and the image indicating the desired viewpoint I_3. Both images have 3 or 4 channels depending on the format (alpha channel) and are in the same resolution. In the following discussion, we use the image size of 1024^2 for the introduction of network architectures. They are concatenated to a tensor of size 1024^2 with 6 or 8 channels. Blocks of Convolutional-BatchNorm-ReLU layers are used in combination to reduce the size of input images iteratively.

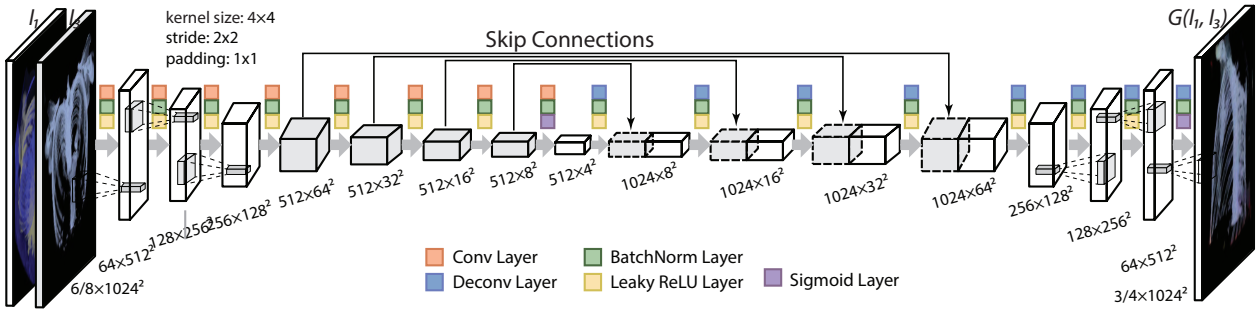

Figure 3: Network architecture of the generator, including encoding and decoding parts. (De)Convolutional layers together with other layers, represented using colored squares, construct the whole network architecture.

As described previously, after passing each layer block, the width and height of tensors will reduce by a factor of 2. At the same time, the number of channels increases. To enable a simple configuration, we choose a multiplier, 64 in our current implementation, and let the number of output channels of each layer blocks be a particular factor time this multiplier. The factors are set to 1, 2 and 4 for the first three layers, and kept at 8 afterward to avoid being too large in model parameters. The configuration mentioned above determines the level of abstraction implicitly. The computation time is also impacted by the complexity of the configuration. In our experiments, we set the number of layer blocks to 8 after several rounds of testing. With such setting, when the input images are 1024^2, as shown in Figure 3, the encoding tensors have a size of 512×4^2.

In the decoding process, Deconvolutional-BatchNorm-ReLU blocks are used to up-sampling images from the encoding results. In contrary to the Convolutional-BatchNorm-ReLU blocks, the width and height are doubled each time. The number of channels starts reducing after several blocks. In the final layer block, the number of output channels is set to 3 or 4, representing the RGB or RGBA channels, and Sigmoid activation $f(x) = \frac{1}{1+e^{-x}}$ is used to obtain values in $[0, 1]$ for each channel. Therefore, we can finally obtain a synthesized image as the output of the generator. For the detail of the model parameter setting, please refer to the appendix.

One special configuration of the generator is the skip connections. As reported in recent works about CNNs [12, 32], skip connections can significantly improve the performance of networks. In classical networks, connections only exist in neighboring layers. However, skip connections will connect layers which are not connected previously, shown as gray blocks and connecting lines in Figure 3. If without skip connections, limited information can be preserved in the encoding tensors of size 512×4^2, which is unfavorable for the image synthesis in our situation. With the assistant of the skip connections, our network can directly transfer important features of the inputs extracted from the encoding part to the decoding part. Also, these features are in different levels of abstractions, which could significantly improve the synthesis quality.

In the discriminators, the input could be a single image of 3 or 4 channels, or two images concatenating to a tensor, depending on the three discrimination tasks we defined previously. Convolutional-BatchNorm-ReLU blocks are used to transform the input images to a small tensor conveying extracted features. These features are used to calculate the cost functions of their discrimination tasks.

Loss functions are defined for both the generator and the discriminators. As lots of equations involved in the definition, more detailed description is given in the appendix. Generally speaking, optimizing the cost functions of discriminators will increase its performance of corresponding discrimination tasks. The generator becomes more skilled to fool the discriminators and produces images closer to the ground truths.

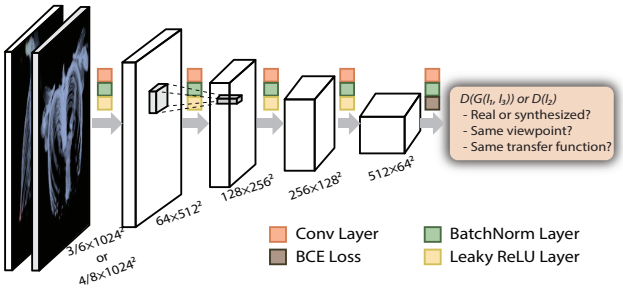

Figure 4: Network architecture of the discriminator. Which takes single image or a couple of images with 3 or 4 channels as input, and projects if the image or the image pair to `true` or `false`.

3.4 Training and Validating

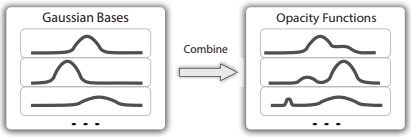

Figure 5: We generate gaussian bases with random mean and deviation; then combine several bases to construct opacity functions

To obtain training samples for our model, we generate rendering images with certain rules. At first, we tune several opacity functions for the volume data. As illustrated in Figure 5, the opacity functions come from combinations of several bases, each of which is a gaussian with random mean and standard deviation in the value range. Some base with specific value range encodes special part in the volume, e.g. the bone part in the chest simulation, which we will use the word feature to refer to. We sampled a large number of opacity Gaussian bases to cover the possible features in the volume. We randomely choose several bases to construct a opacity function. Compared to the sampling strategy proposed by Berger et al. [2], our strategy ensures that data features are not missing. Typically, there are expected to construct hundreds of different opacity functions. Then random colors are assigned to the mean values of each Gaussian peak The whole color functions are obtained through linear interpolation. Opacity and color functions together make different TFs. At last, the volume is colored with TFs, and rendered with sampled different viewpoints to obtain images. In our experiment, the viewpoint is a tuple representing longitudes and latitudes in a sphere, whose radius could vary to indicate viewing distance. Considering the vast parameter space of TFs and viewpoints, we

randomly sample 10,000 pairs of rendering images for training to balance the training time and model performance.

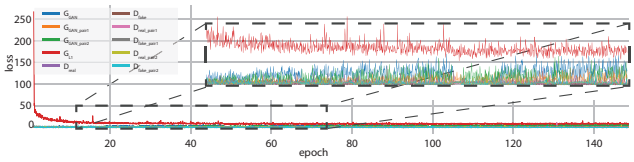

Figure 6: Loss curve of the training process. G_{L1} loss drops quickly after about ten epochs and becomes stable. Other loss terms, although smaller values indicate better performance, form competing for balance in the game framework.

In the training process, the optimizer calculates the sum of losses with a specific set of training samples and derives its gradients on model parameters. The learner then updates parameters based on gradients with certain strategies. The loss is expected to be converged after multiple iterations. With the highly developed deep learning frameworks, we no longer need to implement the backward propagation algorithm for solving the optimization problem with defined loss functions, and we only need to select suitable optimizer and parameter learner.

For the optimizer, mini-batch gradient descent is chosen, in which only a small subset is chosen for total loss calculation. Compared to stochastic gradient descent, where only one randomly picked example is chosen in each iteration or full-batch approach, mini-batch is able to acquire a quick and robust convergence. However, as our model size is relatively large, a very large batch size is not preferred considering the limited GPU memory. In our training process, the batch size is set to 4 to balance convergence and training time. The number of epochs is set to 200, i.e., the model goes through the whole training set 200 times. In our test, the model would get convergence in 20 epochs. Together with mini-batch gradient descent, Adam (Adaptive Moment Estimation) [20] is chosen as the learner for model parameters because of its recognized good performance. Adam can dynamically adjust the learning rate based on the scale of the gradients. It has shown excellent adaptation of learning rate in different applications.

Figure 6 shows the loss curves in one run of model training. From the plot, the L1 loss measuring the difference between the synthesized image and the ground truth one drops quickly after about ten epochs and becomes stable in about 20 epochs. For other loss terms, although smaller values indicate better performance, they form competing relations in the game framework. So they also reach balance states after tens of epochs. In summary, the current setting is sufficient to achieve a relatively stable convergence after about tens of epochs.

We further create a set of rendering images for validation. These validation images are rendered with totally new transfer functions and random viewpoints under the same volume data. We show the synthesis performance, e.g., time cost and quality in the following section.

3.5 Implementation and Deployment

Our model is implemented using PyTorch[1]. The model is deployed on a symmetric multiprocessing node, equipped with 8 NVIDIA M6000 graphics cards, with 3072 CUDA cores and 12GB memory each. The data is stored on a local disk with 2TB capacity.

4 Interactive System

To efficiently utilize our approach, we further develop an interactive system, as shown in Figure 7. The learning model is embedded in the system to synthesize images with goal images provided.

[1]pytorch.org

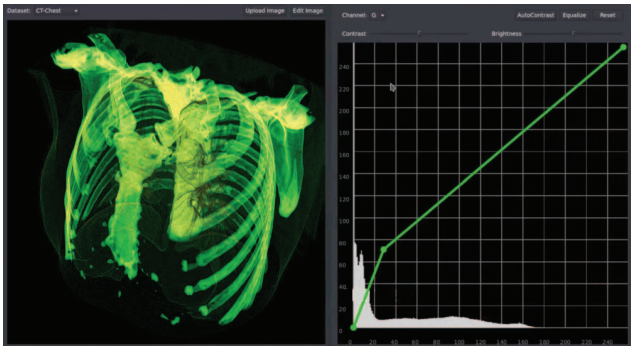

Figure 7: Interface of our interactive system. In the left is the volume exploration view, it supports choosing a dataset and uploading a goal image. Direct 3D-navigation is supported here. In the right is the image-level editing view, contrast, brightness and color curves are allowed to be edited int the editing view.

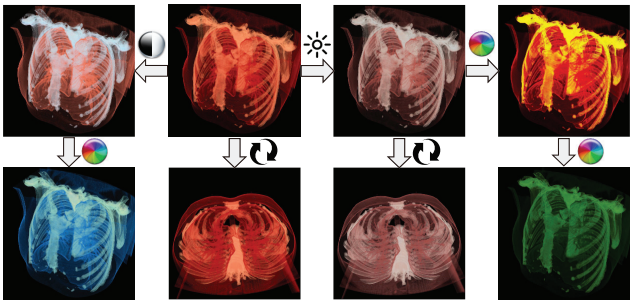

Figure 8: Typical exploration paths enabled by our interactive system. Applying 3D-Navigation. changing contrast. brightness ajusting. changing the RGB curves of the uploaded images.

From the user's perspective, the whole workflow is simple and intuitive, as shown in the left part of Figure 2. In the beginning, the user can choose one trained volumetric data, and upload a corresponding rendering image of this data to inform the system the desired transfer function effect. Users are also allowed to directly edit the uploaded image with image-level operators. After the input image is specified, the system can generate corresponding images with the learning model. The user can start performing navigation operations in the display area, such as rotation, panning, and zooming. Underlying the system, for every frame, an image indicating the current viewing parameter is first generated with a rendering pipeline. The goal image and the pre-rendering image are then fed to the trained model to synthesize the new image with the desired transfer function and viewing parameter. The synthesized image is shown in the interface.

To enable editing on the uploaded images, our system embeds an interface for users to conduct basic image operators. Currently, adjustments of contrast, brightness and color curves have been supported. In Figure 8, several synthesized images after users' interactions and edits are shown. In the original uploaded image, the color of bones is a little bit red, which makes it difficult to be distinguished from lungs and skins. Users can augment the image by manual edits by adjusting either contrast and brightness or color curves. After either modification, Bones and lungs become perceptually more distinct after adjustments. The user can make the further significant color change by adjusting the color curves. If the user expects more sophisticated editing, other out-of-the-box tools, like Photoshop or GIMP, can be used to edit images, and then be uploaded to our system for exploration. However, of course, our system can not always

perform well for arbitrary edited images, as the target image must also consistent with the underlying volume features. This constraint is set by the fact that our training images are all generated from the volume rendering pipeline with explicit transfer functions.

A critical issue of the interactive system is the computation performance. As shown in Table 1, synthesizing images of resolution 1024^2 needs about 0.4 seconds, which could make the interaction less fluent. To provide more interactive behavior of the system, we synthesize low-resolution images during the interactions and high-resolution ones when interactions are complete. If images of size 256^2 are used as the intermediate results, the computation efficiency can reach more than 10 FPS, which is sufficient for a smooth exploration.

5 Results

We evaluate the results of our model from two aspects: the synthesis quality and time/storage costs under different datasets, image formats, and hyper-parameters.

5.1 Different Datasets

Three datasets of different types are used for the evaluation: CT-Chest dataset, Combustion Simulation dataset, and Carp dataset. Their spatial resolutions are $384 \times 384 \times 240$, $480 \times 720 \times 120$, and $256 \times 256 \times 512$ respectively. Figure 9 shows the results, in which each row contains the images indicating the goal image I_1, the viewpoint image I_3, the synthesized image $G(I_1, I_3)$, the ground truth image I_2, and the difference image between the synthesized one and the ground truth one $|I_2 - G(I_1, I_3)|$. Generally speaking, the shapes of volume data and the colors and transparencies are synthesized quite well. The synthesized images show very high photorealistic quality and perceptual fidelity. If only given synthesized images, regular users could hardly identify whether the images are generated from learning models or a rendering pipeline. If compared with the ground truth images, we can observe subtle differences with careful investigation, as shown in the difference images. The differences are mainly scattered in the boundary of the data features. Overall, there are no significant reconstruction errors in the synthesized images.

Also, we employ PSNR (Peak Signal-to-Noise Ratio) to measure the quality of synthesized images. Our synthesis obtains PSNR values of more than 2 dB. For some testing cases, they can reach 30 dB and higher. As common lossy image and video compression algorithms have PSNR of 30-50 dB [35], the quality of our algorithm shows close or comparable accuracy compared to them. In our case, the synthesized images will not lead to severe misunderstanding in most situations, where extreme precision is not in demand. On the other hand, the small loss of accuracy is the trade-off to achieve the capability of goal-oriented volume visualization from images.

Table 1: Statistics of different datasets and image sizes. T_s the synthesizing time (s), T_r the pre-render time (s), T_t total time of both (s). S_m the size of the model (MB), S_d the size of volumetric data (MB), S_I the size of Image (pixel * pixel). PSNR Peak signal-to-noise ratio (dB).

Data	S_d	S_I	T_s	T_r	T_t	FPS	PSNR	S_m
Combustion	159	128^2	0.012	0.036	0.077	13.0	22.3	193
		256^2	0.025	0.043	0.090	11.1	26.4	193
		512^2	0.086	0.065	0.151	6.62	27.3	241
		1024^2	0.278	0.087	0.365	2.74	24.5	241
CT-Chest	34	1024^2	0.293	0.085	0.378	2.65	24.9	241
Carp	33	1024^2	0.301	0.083	0.384	2.60	28.8	241

5.2 Image Masking Effects

In our initial experiments, we have found that the masking effects of input images can influence the synthesis quality significantly, especially in situations where the volumetric objects only occupy a small portion of the image. In the following, we use the Carp dataset to demonstrate the effects of choosing RGB or RGBA channels. In the Carp dataset, if we keep the viewport to be square, more than half of the image is occupied by the background. When training with these images, the L1 loss measuring reconstruction error drops quickly to a low value and then keeps relatively stable. This phenomenon makes the generator with less moment to optimize, due to small gradients, which can be considered as a representation of *gradient vanishing*. Since the problem mainly comes from the input images themselves, we try to tackle it with different input configurations.

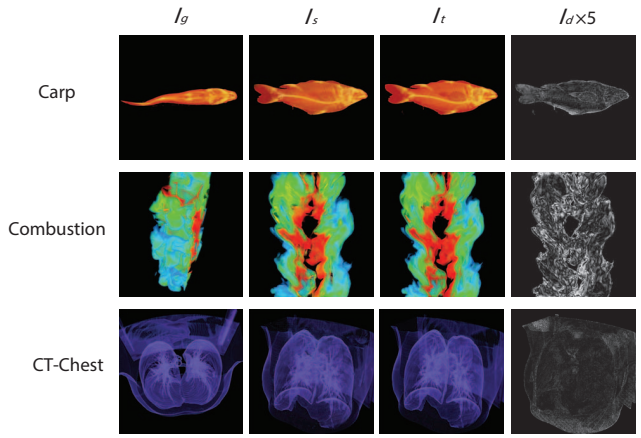

Figure 9: Synthesis results for different datasets. I_g the goal, I_s the synthesized, I_t the ground truth, I_d the difference between synthesized and ground truth. We show the $I_d \times 5$ due to the insignificant value of I_d. For the biomedical data, there is a clear edge in different parts; the artifacts are possible to occur at the edge of each component. For the combustion data, the value is more continuous that the artifacts are evenly distributed in the image.

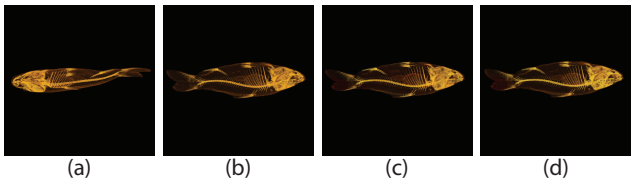

Figure 10: Results of Carp dataset with different configurations. (a) The goal image, (b) the ground truth image, (c) the synthesized images with RGB channels as inputs and the weight for L1 loss λ set to 1000, and (d) the synthesized images with RGBA channels as inputs.

Three different configurations are tested in our experiments: using RGB formats, clipping the image, and using RGBA formats with the alpha channel indicating the background. In Figure 10, we show (a) the goal image, (b) the ground truth image, (c) the synthesized images with RGB channels as inputs the weight for the L1 loss λ set to 1000, and (d) the synthesized images with RGBA channels as inputs. If using RGB formats and λ is set to a low value 100, the result gives a PSNR of 29.4 dB. Instead, the λ increases to 1000, the PSNR improves significantly to 36.1 dB, and the result becomes quite good as shown in (c). As for the clipped input, one side-effect is that the clipped image needs to resize to match the model. Due to the blurred effects after resizing, the quality of synthesized images does not improve (PSNR 27.6 dB). In the third configuration, with RGBA formats, it gives a vast improvement as shown in (d), reaching a PSNR of 38.0 dB. The information of the alpha channel makes the model focus on the object part without interference from

the background. Our experiments are consistent with the practice in image-related applications in the machine learning community, where certain algorithms are employed to detect the background to improve the task performance. In our task, since the background is in pure color, it can be directly identified with a proper alpha channel setting. With the above experiments results, we decide to use a relatively large λ, 1000 in this work, and RGBA formats as inputs whenever possible.

5.3 Different Hyper-Parameters

Several hyper-parameters affects the performance directly, such as the image sizes of inputs and outputs, the number of layers in the network, and the number of channels in each layer, etc. However, modifying hyper-parameters related to the network architecture leads to significant changes in the final performance, which makes them unmatched for comparison. Therefore, we do not exhaust every combination of hyper-parameters and report their performance, as some of them are unmatched in performance. We mainly test the model performance under different input/output sizes of images, as they influence the computation efficiency of our interactive system. When the image size changes, the number of layer blocks have to be adjusted accordingly to keep the quality of the synthesized results. For example, if the input image has size 256^2 and eight layer blocks are still used in the encoding process, then the encoding tensor is as small as 512×1^2 which conveys insufficient information from the encoding part to the decoding part. In such cases, the number of layer blocks is reduced accordingly. However, as unmatched numbers of layer blocks and image sizes could lower synthesis quality significantly, we do not test this factor separately. As Figure 12 demonstrates, the skip layers can improve the image construction ability. Besides, the including of the multidiscriminators can have lower generator loss when training.

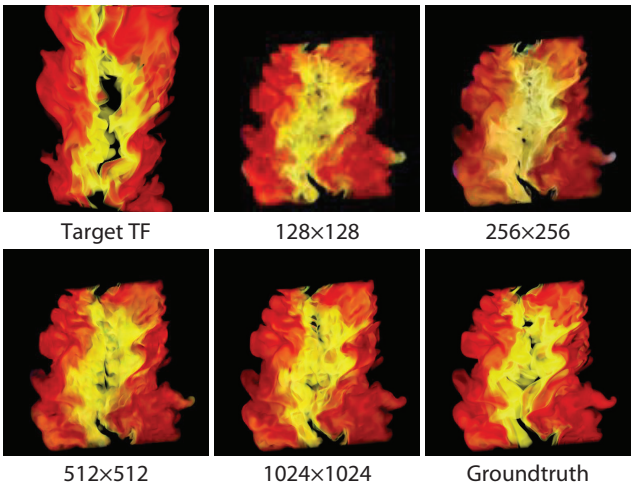

Figure 11: Quality of synthesized images with different sizes of inputs/outputs.

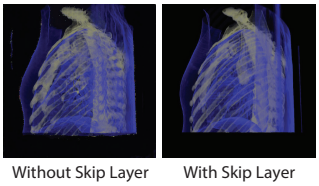

Figure 12: Left without skip layers, right with skip layers

We use Combustion simulation data for this experiment, where variable `mixfrac` is used. The total size of this variable is 159MB with 4-byte floating numbers for storage. In Figure 11, the synthesized images under different sizes are presented, as well as the goal image and the ground truth image. It is evident that results of 128^2 and 256^2 are not as good as those of large sizes. In Table 1, the PSNR values for these small-size inputs are not low. It indicates although lots of details are lost, the main reason is their resolutions instead of the approach. The overall shapes of the volume data are restored relatively well, and colors and transparencies are kept in general. For the result of 512^2, the quality becomes satisfactory. Typical human users could have difficulty to distinguish whether the image is synthesized by our learning model or generated from a volume rendering pipeline. There is no surprise that the resolution of 1024^2 gives the highest quality, not only for the sharpness of synthesized details but also for the accuracy shown as the PSNR value. A careful comparison could show that, with higher resolutions, the details in the boundary parts are synthesized much better than 512^2.

In the table of Figure 9, the time costs, and storage overheads are listed. The synthesis time increases as the image size becomes larger, which is roughly proportional to the image sizes, i.e., the number of pixels. While, the total time considering the pre-render time is nearly doubled, when the image size is doubled. According to the testing results, in our interactive system, we display synthesized images of 1024^2 only when interaction is complete, and 256^2 when interactions are ongoing.

As for the storage overheads, 512^2 and 1024^2 cost same storage, i.e., 241 MB. This is because their network architectures are the same, where 8 (do)convolutional layer blocks are used. Although the sizes of input/output images and intermediate tensors are not the same, the numbers of network parameters do not change. As for 128^2 and 256^2, since using eight layer blocks leads to too small encoding tensors, seven blocks are used instead. The model sizes reduce accordingly. One interesting fact is that the time/storage costs of our model only depends on the resolution of input/output images, rather than the original volume data. It means, under current network architectures, the model sizes are invariant if the rendering image sizes are fixed to 1024^2, no matter what the spatial resolution of the volume data is. The property could be beneficial if the original volume data has a very high spatial resolution.

As for training time, currently, for images of resolution 1024^2, with four cards, the model needs about 2 hours to get convergence in 20 epochs.

5.4 Input Images from Literatures

At last, we test our model with input images from previous works. Although several datasets have been utilized for transfer function design approaches, most of them are not entirely suitable for testing, either for low resolution or overlaid annotations. In Figure 13, we take screenshots of figures (a) [40] and (c) [18] from the manuscripts and feed them to our model as RGB images, which is pre-trained using the random sampled renderings of the same volume data. Note that, although we have already struggled to take high-resolution screenshots of these figures, they still suffered from blurriness. The actual resolution may be lower than 512^2, which affects the quality of our synthesized images significantly. Nonetheless, our approach can still generate generally good results, as shown in (b) and (d). For the Carp dataset, the overall structures and details are synthesized pretty accurately, especially for the bones and organs. Only small color shifts observed in the synthesized images. As for the Engine dataset, the structures are shown quite well, even for some detailed features.

Note that we do not have the underlying transfer functions of the test images, and are unable to involve them in the training set. From the results, our model generally shows the adequate capability of generalization for images with unseen transfer functions. One future

direction is to study how to keep high reconstruction quality even when the input images are in low resolutions or blurred, and with more sophisticated transfer functions.

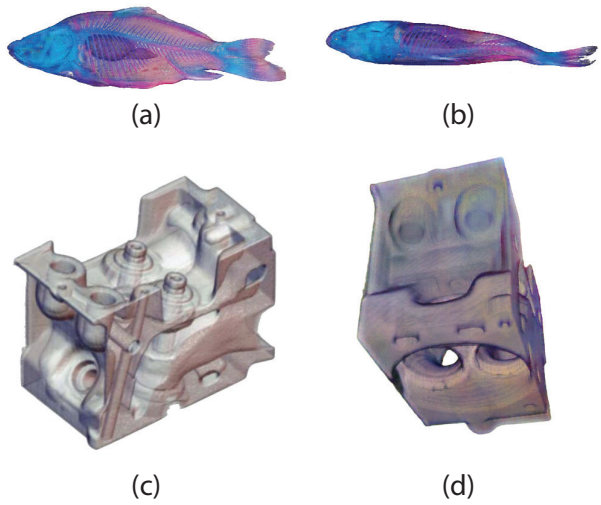

Figure 13: Results of Carp and Engine dataset, (a) and (c) the rendering from existing literatures (Courtesy of [18, 40]). (b) and (d) the synthesized images under different viewpoints.

6 Discussion

In this work, we employ deep learning models to realize the 'reverse' volume rendering pipeline. We have presented our method, the interactive system and several results in previous sections. In the following, we discuss several related issues.

6.1 Image Quality

One of the remaining issues is the quality evaluation of synthesized results. In the volume rendering process, the mappings of the original data are very complicated, and the information loss from 3D volume data to 2D images is inevitable. The reconstruction error of extracting information from rendering images is not possible to avoid entirely. However, from the results presented above, we believe our learning models synthesize images with very high fidelity for human eyes. We show the difference images between our synthesized images and the ground truth ones. There are no significant structural errors between synthesized and ground truth images, only some slight differences in details. Besides, we utilize PSNR to measure the quality of synthesized images. Our synthesized images show PSNR values that are close to and sometimes can reach beyond those of common lossy image and video compression algorithms. Such results also support that the images generated by our models will not make severe mistakes or misunderstanding when exploring the volumetric data.

From all these aspects above, we think the current quality is tolerable if not in precision strongly-demanded situations. In exchange, our approach can achieve more than a traditional rendering pipeline, that is the direct correspondences between the desired rendering effects and rendering images.

6.2 Limitations

As deep learning techniques are employed in our approach, we suffer certain drawbacks that are shared by them.

One is the training time of the learning models. The reason for costing several hours comes from the great large resolution of input and synthesized images. In classical applications of GANs, the generated images are usually smaller than or equal to 256^2, which is not feasible for our implementation of volume visualization.

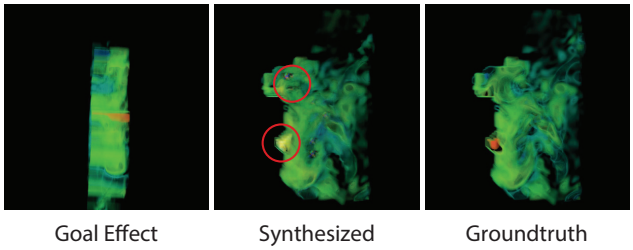

Figure 14: The failure case generated by our model. There are wrong colors and artifacts in the red circles of the synthesized image.

Current editing method is not allowed to be too different from those generated from the regular rendering pipeline. Images with unusual styles may cause the model to fail to synthesize corresponding results because the training data comes from the volume rendering pipeline. Some rendering images from previous literature suffer from such restrictions, due to sophisticated techniques they used or unusual effects. More training samples could add to the training set to broaden its usage scenarios.

Failure case may occur when the input goal effect have an unusual viewpoint that it did not present the information needed in the synthesized image. The failure case can be demonstrated by Figure 14, with wrong colors and artifacts. The viewpoint of the goal-effect image is on the side, which did not provide the information on the front side. In this case, it's difficult to ask the model to generate the right color.

Besides, we've tried to train a generic model for not only a simple dataset. However, the complexity in 3-dimension data makes the goal hard to achieve. In the next step, we will extend the model into a larger range for a class of volume data with certain simalarities.

7 Conclusion and Future Works

In this work, we employ deep neural networks, specifically a combined usage of convolutional neural networks (CNNs) and generative adversarial networks (GANs), to achieve the task of 'reversely' volume exploration directly from images. Our approach enables 3D navigation operations on provided goal images for exploration of corresponding volumetric data by direct image synthesis based on the extracted information from inputs. Our learning model works not only for provided images but also for images with image-level edits. The presented results have successfully demonstrated the effectiveness of our proposed approach, and the potential of deep neural networks in volume rendering-related tasks.

In the future, we would like to work further to improve the precision and generalization ability of our approach, including more extreme camera positions, more various and sophisticated rendering styles, and multiple datasets. Also, our pipeline can be a method to deliver data, especially in some scenarios where the volumetric data is not open-sourced but the given resolution of exploration is allowed. Besides, as the model size is invariant with the volume size, with some learning model compression method, our approach can be a kind of storage-saved method to deliver data.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. This work is supported by NSFC No. 61872013 and 61672055. This work is also supported by PKU-Qihoo Joint Data Visual Analytics Research Center. The authors would like to thank Prof. Junping Zhang from Fudan University for his valuable suggestions.

REFERENCES

[1] H. Akiba and K. Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proc. of Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 115–122, 2007.

[2] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[3] C. D. Correa and K. Ma. Size-based transfer functions: A new volume exploration technique. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1380–1387, 2008.

[4] C. D. Correa and K. Ma. The occlusion spectrum for volume classification and visualization. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1465–1472, 2009.

[5] C. D. Correa and K. Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Trans. Vis. Comput. Graph.*, 17(2):192–204, 2011.

[6] I. Fujishiro, T. Azuma, and Y. Takeshima. Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In *Proc. of IEEE Visualization*, pages 467–470, 1999.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[9] H. Guo, N. Mao, and X. Yuan. WYSIWYG (What You See is What You Get) volume visualization. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2106–2114, 2011.

[10] H. Guo, H. Xiao, and X. Yuan. Scalable multivariate volume visualization and analysis based on dimension projection and parallel coordinates. *IEEE Trans. Vis. Comput. Graph.*, 18(9):1397–1410, 2012.

[11] H. Guo and X. Yuan. Local WYSIWYG volume visualization. In *Proc. of IEEE Pacific Visualization Symposium (PacificVis)*, pages 65–72, 2013.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[13] T. He, L. Hong, A. E. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In *Proc. of IEEE Visualization*, pages 227–234, 1996.

[14] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of the International Conference on Machine Learning, ICML*, pages 448–456, 2015.

[16] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5967–5976, 2017.

[17] C. R. Johnson and J. Huang. Distribution-driven visualization of volume data. *IEEE Trans. Vis. Comput. Graph.*, 15(5):734–746, 2009.

[18] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Volume Visualization, 1998. IEEE Symposium on*, pages 79–86. IEEE, 1998.

[19] G. L. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proc. of IEEE Symposium on Volume Visualization*, pages 79–86, 1998.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] J. Kniss, G. L. Kindlmann, and C. D. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 8(3):270–285, 2002.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017.

[25] A. Lu and D. S. Ebert. Example-based volume illustrations. In *Proc. of IEEE Visualization*, pages 655–662, 2005.

[26] J. Marks, B. Andalman, P. A. Beardsley, W. T. Freeman, S. F. Gibson, J. K. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. M. Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proc. of Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 389–400, 1997.

[27] H. Pfister, W. E. Lorensen, C. L. Bajaj, G. L. Kindlmann, W. J. Schroeder, L. S. Avila, K. Martin, R. Machiraju, and J. Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001.

[28] T. M. Quan, J. Choi, H. Jeong, and W. Jeong. An intelligent system approach for probabilistic volume rendering using hierarchical 3d convolutional sparse coding. *IEEE Trans. Vis. Comput. Graph.*, 24(1):964–973, 2018.

[29] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[30] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016.

[31] C. Rezk-Salama, M. Keller, and P. Kohlmann. High-level user interfaces for transfer function design with semantics. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1021–1028, 2006.

[32] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

[33] S. Röttger, M. Bauer, and M. Stamminger. Spatialized transfer functions. In *Proc. of Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 271–278, 2005.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[35] A. Said and W. A. Pearlman. An image multiresolution representation for lossless and lossy compression. *IEEE Transactions on Image Processing*, 5(9):1303–1310, 1996.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[37] I. Takanashi, E. B. Lum, K. Ma, and S. Muraki. ISpace: Interactive volume data classification techniques using independent component analysis. In *Proc. of Pacific Conference on Computer Graphics and Applications*, pages 366–374, 2002.

[38] F. Tzeng, E. B. Lum, and K. Ma. A novel interface for higher-dimensional classification of volume data. In *Proc. of IEEE Visualization*, pages 505–512, 2003.

[39] F. Tzeng, E. B. Lum, and K. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Trans. Vis. Comput. Graph.*, 11(3):273–284, 2005.

[40] Y. Wu and H. Qu. Interactive transfer function design based on editing direct volume rendered images. *IEEE Trans. Vis. Comput. Graph.*, 13(5):1027–1040, 2007.

[41] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2Image: Conditional image generation from visual attributes. In *Proc . of European Conference on Computer Vision (ECCV)*, pages 776–791, 2016.

[42] H. Zhang, T. Xu, and H. Li. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 5908–5916, 2017.

[43] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017.