

# Differentiable Design Galleries: A Differentiable Approach to Explore the Design Space of Transfer Functions

Bo Pan, Jiaying Lu, Haoxuan Li, Weifeng Chen, Yiyao Wang, Minfeng Zhu, Chenhao Yu, and Wei Chen

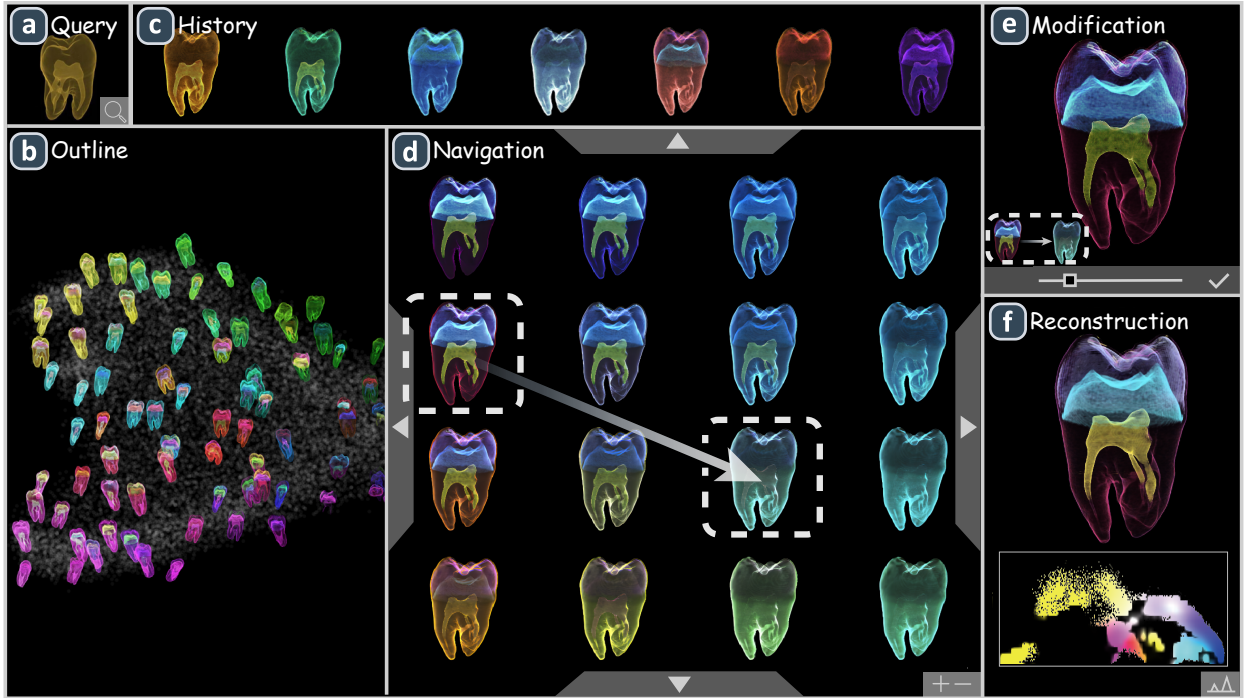


Fig. 1: The interface of Differentiable Design Galleries enables users to intuitively explore the design space of transfer functions. (a) The Query View accepts an exemplar as a query to find similar designs within the design space. (b) The Outline View provides an overview of the design space. (c) The History View keeps a record of user exploration. (d) The Navigation View allows users to navigate through the subspace constructed from selected designs. (e) The Modification View specifies a semantic direction for modifying a target design. (f) The Reconstruction View presents the reconstructed transfer function based on the target design.

**Abstract**—The transfer function is crucial for direct volume rendering (DVR) to create an informative visual representation of volumetric data. However, manually adjusting the transfer function to achieve the desired DVR result can be time-consuming and unintuitive. In this paper, we propose Differentiable Design Galleries, an image-based transfer function design approach to help users explore the design space of transfer functions by taking advantage of the recent advances in deep learning and differentiable rendering. Specifically, we leverage neural rendering to learn a latent design space, which is a continuous manifold representing various types of implicit transfer functions. We further provide a set of interactive tools to support intuitive query, navigation, and modification to obtain the target design, which is represented as a neural-rendered design exemplar. The explicit transfer function can be reconstructed from the target design with a differentiable direct volume renderer. Experimental results on real volumetric data demonstrate the effectiveness of our method.

**Index Terms**—Transfer function, direct volume rendering, deep learning, generative models, differentiable rendering

## 1 INTRODUCTION

Direct volume rendering (DVR) is a powerful and effective visualization technique for exploring and studying volumetric data, which is widely used by scientists, engineers, physicians, and artists in various applications [7]. The expressiveness and flexibility of DVR stem from the large design space of the transfer function (TF). Specifically, a carefully adjusted transfer function helps to create a meaningful and informative visual representation of complex data by mapping scalar values to visual properties, such as color and opacity [27].

However, finding an appropriate transfer function is a non-trivial task [34]. One of the most commonly used tools is transfer function design widgets, with which users can define and refine the shape of the target transfer function. While such widgets are powerful and flexible for experienced engineers or scientists, the design process is usually time-consuming and unintuitive for inexperienced users: subtle changes

- Bo Pan, Jiaying Lu, Haoxuan Li, Yiyao Wang, Chenhao Yu, and Wei Chen are with the State Key Lab of CAD&CG, Zhejiang University, and Wei Chen is also with the Laboratory of Art and Archaeology Image (Zhejiang University), Ministry of Education, China. E-mail: {bopan | jiangyinglu | lihaoxuan | wangyiyao | ych20 | chenvis}@zju.edu.cn.
- Weifeng Chen is with Zhejiang University of Finance&Economics. E-mail: cwf818@zufe.edu.cn.
- Minfeng Zhu is with Zhejiang University. E-mail: minfeng\_zhu@zju.edu.cn.
- Wei Chen and Weifeng Chen are the corresponding authors.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.  
Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxxx

in the transfer function domain may lead to an unexpected change in the image domain due to the nonlinearity and complexity of the volume rendering process. To address this issue, some researchers propose Design Galleries to provide users a convenient way to explore pre-defined design spaces of transfer functions [10, 31, 35, 36]. Users can directly select a rendered image in the image gallery to obtain the desired transfer function, rather than manually setting one. This approach is more user-friendly and intuitive for non-expert users seeking their desired transfer function. However, several problems limit the usability of Design Galleries: First, when users already have a specific design style in mind, there is no efficient method to query the target design within the design space. Second, it is unclear how to efficiently navigate the high dimensional transfer function design space [19]. Third, it is infeasible for users to modify the rendered images they are interested in within Design Galleries, unless they revert back to using the traditional transfer function widgets. To address the aforementioned problems, we propose Differentiable Design Galleries, a transfer function design approach based on deep learning and differentiable rendering to assist users in exploring the design space of transfer functions. By leveraging a latent design space, users can engage in various types of transfer function exploration, including querying, navigation, and modification, within the image domain without resorting to explicit transfer functions.

The principal challenge in constructing a Differentiable Design Gallery lies in mapping the discrete design exemplars of the design space into a latent design space, which is a continuous manifold representing various types of implicit transfer functions. We tailor a framework comprising three key neural networks, namely an encoder, a generator, and a discriminator, to perform the latent design space generation. The encoder can analyze a design exemplar, typically a DVR image, and map it into a latent code in the latent design space. This latent code serves as a representation of the implicit transfer function linked to the underlying transfer function used to render the design exemplar. To ensure that the latent design space is well-organized for subsequent explorations, we tailor an “imitation task” to train our networks. Once the latent design space is generated, users can use a set of interactive tools to explore this design space and locate their desired transfer function design, which is represented as a neural-rendered design exemplar (generated by the generator of our framework). Finally, our approach can reconstruct the explicit transfer function based on the target neural-rendered design exemplar with the aid of a differentiable DVR renderer.

We summarize our contributions as follows:

- Differentiable Design Galleries, an approach which provides users with an intuitive way to explore the transfer function design space in the image domain.
- A tailored framework to train an encoder that can map the transfer function design space represented by discrete exemplars into a well-organized latent design space.
- A set of interactive tools that help to explore the latent design space and obtain the expected DVR result.

## 2 RELATED WORK

### 2.1 Transfer Function Design

The transfer function is crucial for direct volume rendering to create an informative visual representation of volumetric data. How to design transfer functions and facilitate the transfer function design process has been continuously discussed in the past three decades.

Generally, there are two main categories of transfer function design methods: data-centric methods and image-centric methods [34]. For data-centric methods, the most basic transfer function is to define the mapping of scalar values to visual properties, which is the so-called one-dimensional transfer function. This transfer function is simple and efficient for some scenarios. However, it would be inadequate when features of interest exhibit overlapped intensity or are affected by noise [28]. Different types of multidimensional transfer functions are proposed to better depict volume data. Kindlmann and Durkin [21] presented a histogram volume, which captures the relationship between

the data value and its first and second directional derivatives along the gradient direction, for semiautomatic transfer function generation with user guidance. Then, Kniss et al. [24] extended their work by introducing a set of direct manipulation widgets and a novel dual-domain operation that makes specifying transfer functions more intuitive and convenient. Later, the community proposed a variety of attributes for transfer function design. For example, curvatures [22] can be used to better deliver surface information; approximated size [2] helps locate features with a certain size; visibility [4] emphasizes features’ visibility from different viewpoints; ambient occlusion [3] reveals occlusion relationships.

Although well-elaborated derivatives of original volumetric data may increase transfer functions’ expressiveness, the difficulty of transfer function adjustment and generation also rises [27], which is challenging for non-expert users. A general question shared by those data-centric approaches is: How can we effectively constrain the design space of transfer functions based on our prior knowledge about the data? Tzeng and Ma [40] proposed an iterative self-organizing data analysis technique to find clusters in 2D histogram space. Subsequently, user interaction is simplified to modify these clusters. Other strategies to cluster the feature space are also explored, such as kernel density estimation [30], Gaussian mixture model [43], and valley cell-based clustering [44]. Dimensionality reduction methods are utilized to simplify high-dimensional transfer functions to lower-dimensional space [5, 12, 26]. Semantic-based transfer functions [38] are proposed to aggregate low-level transfer functions for a more intuitive transfer function design process. Ip et al. [17] leveraged information theory to hierarchically cut the 2D gradient intensity histogram into segments.

While the data-centric methods still leave a non-intuitive gap between the transfer function domain and the resulting image, the image-based methods allow users to directly explore the design space from the image side. He et al. [13] proposed a semi-automatic approach that lets users make selections among rendering results. These selections supervise a genetic algorithm to generate better transfer functions. Guo et al. [11] demonstrated a WYSIWYG volume exploration framework that allows users to modify transfer functions by directly editing the rendering result using the provided tools. Wu and Qu [50] proposed an interactive transfer function design approach by blending different direct volume rendered images. Design Galleries [10, 31, 35, 36] try to generate a sufficient number of samples to span a transfer function design space, allowing users to explore and choose among the presented possibilities. However, how to provide users with a finer degree of control when they are not satisfied with the initially generated images within Design Galleries remains an unsolved problem. How to efficiently explore the high dimensional transfer function design space represented by the Design Galleries is also under discussion.

By leveraging recent advances in deep learning and differentiable rendering, our Differentiable Design Galleries approach enables users to perform query, navigation, and modification within the image domain to achieve their desired transfer function design.

### 2.2 Deep learning for Scientific Visualization Generation

Deep learning techniques bring crucial benefits to the SciVis research community. In 2022, Wang and Han [42] conducted a thorough survey about the application of deep learning techniques for scientific visualization. Here we review the works that utilize deep learning for scientific visualization generation.

Berger et al. [1] made the first attempt to use a generative network with an explicit texture transfer function as input to replace traditional DVR renderers. Their generator can support transfer function sensitivity analysis and provide an overview of possible 1D opacity transfer functions, which is helpful in guiding users’ volume exploration processes. He et al. [14] used a generator to directly learn a mapping from simulation and visualization parameters to the rendering results, which supports efficient analysis of the underlying ensemble simulations. Hong et al. [15] leveraged a DVR image as the implicit transfer function input for a generator in their pipeline so that users could explore the given volumetric data without an explicit transfer function. To improve the quality of the generated visualization, different forms

of auxiliary information are proposed and provided to the networks. Weiss et al. [46] fed low-resolution normal and depth fields to the neural network to obtain high-resolution isosurface maps. Weiss et al. [47] proposed a neural rendering framework that consists of one sub-network to generate a sparse adaptive sampling structure and another sub-network to generate the high-resolution image based on the sampling result. Some works propose to integrate the neural network with a differentiable rendering pipeline. Weiss and Navab [45] tried to integrate the neural network into different stages of the DVR process so that the rendering process can be optimized from manually adjusted reference images. Weiss and Westermann [48] presented a memory-efficient differentiable DVR renderer and showed the potential of integrating the renderer with different loss terms and neural networks.

Our work extends this line of research by leveraging neural rendering and differentiable rendering to help users explore the design space of transfer functions.

### 3 OVERVIEW

The overview of our approach is presented in Fig. 3, which consists of three main stages. First, given specific volumetric data and a pre-defined transfer function design space, we generate design exemplars using a traditional DVR renderer, with randomly sampled transfer functions within the design space. Then we train an encoder and a generator to generate a latent design space based on those design exemplars. The training task is specifically tailored such that the generated latent design space is well organized (Sec. 4). Second, we propose a set of tools to help users explore this latent design space and integrate them into an interactive interface (see Fig. 1). With our interactive interface, users can (1) Gain a rough overview of the design exemplars within the design space, just like in traditional Design Galleries. (2) Use an image to query a similar design within the latent design space. (3) Navigate the latent design space with intuitive operations. (4) Make semantically meaningful modifications on a target design without resorting to transfer function widgets (Sec. 5). Third, when users are satisfied with a certain design (represented as a neural-rendered design exemplar), we leverage a differentiable DVR renderer to reconstruct the desired explicit transfer functions in texture form (Sec. 6).

### 4 LATENT DESIGN SPACE GENERATION

In this section, we will elaborate on the framework for latent design space generation. The key generation process involves finding a suitable mapping function  $F$  that transforms discrete design exemplars into latent codes. To enable effective utilization of the latent design space in downstream applications, we aim for the mapping function  $F$  to possess the following properties:

- (1) Similar design exemplars should be mapped to latent codes that are close together in the latent design space.
- (2) The mapping should be invariant with respect to the viewport of the design exemplars. For instance, if two design exemplars have different viewports but share the same underlying transfer function, they should be mapped to the same latent code.

We propose utilizing an encoder network to learn the mapping function  $F$ , leveraging the expressive capabilities of neural networks as function approximators [16]. The training task for the encoder is designed as an “imitation task”, resembling the inverse engineering

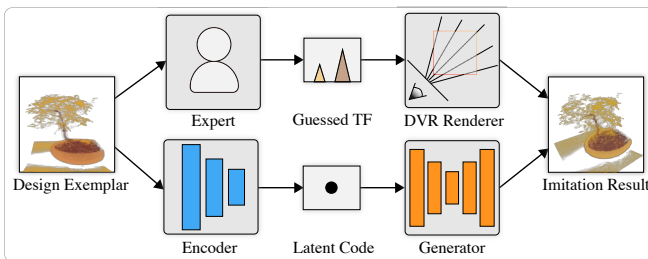


Fig. 2: Comparison of the processing steps for the “imitation task” between our model and a human expert.

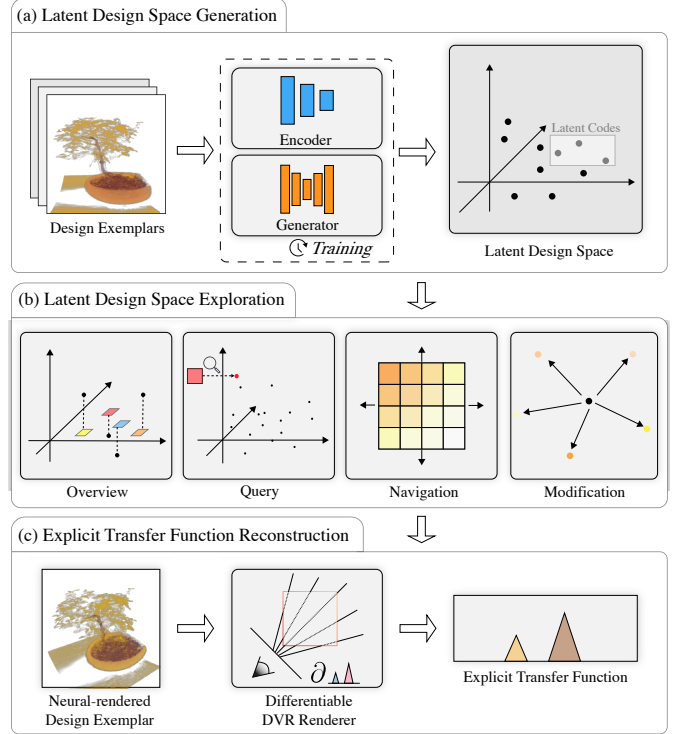


Fig. 3: Overview of our approach. First, we generate the latent design space based on sampled design exemplars. Second, we explore the latent design space using a set of interactive tools. Third, we reconstruct the explicit transfer function from a neural-rendered design exemplar.

process of a DVR expert. As shown in Fig. 2, given a DVR image of volumetric data as a design exemplar, a DVR expert can analyze the features shown in the design exemplar and guess the underlying transfer function. If the guess is good enough, the image rendered by a DVR renderer with this guessed transfer function should closely resemble the design exemplar. Similarly, the encoder tries to analyze the features shown in the design exemplar and map it into a corresponding latent code. If the mapping is successful, the image rendered by a generator with this latent code should be similar to the design exemplar.

#### 4.1 Overview of the framework

Fig. 4 illustrates the overview of our framework. During training, there are three sub-networks: the encoder, the generator, and the discriminator. The input of the encoder is a design exemplar. The design exemplar is rendered by a DVR Renderer with viewpoint  $V_{exp}$  and transfer function  $T_{exp}$ . The encoder analyzes the design exemplar and maps it into a latent code. The input of the generator is this latent code and a viewport image. The viewport image is rendered by a DVR renderer with viewpoint  $V_{view}$  and an auxiliary transfer function  $T_{view}$  to provide viewport information to the generator. The generator is expected to generate an image that resembles the result of rendering by a DVR renderer with viewpoint  $V_{view}$  and transfer function  $T_{exp}$ . To facilitate the training of the encoder and the generator, we use both pixel-wise comparison and a discriminator to compute the loss between the neural-rendered image and the ground truth image. The discriminator is trained to classify whether a given image is from a generator or a DVR renderer. The classification result can be used as the adversarial loss [9]. Previous works [1, 14, 15, 18] have shown that this adversarial loss can alleviate the over-smooth problem caused by pixel-wise loss and improve the perceptual quality of the rendered images. Once the training is complete, we no longer need the discriminator. We can use the encoder and the generator separately or jointly for different downstream tasks.

#### 4.2 Encoder

The architecture of the encoder is shown in Fig. 5 (a), which takes a design exemplar as input and outputs a 64-dimensional vector, which is



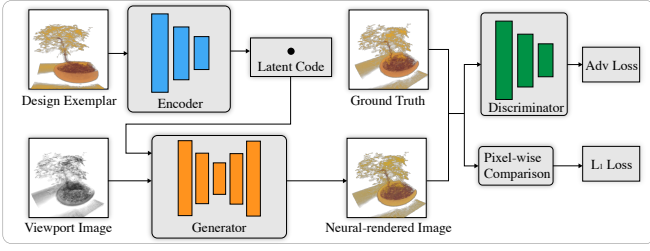


Fig. 4: The framework for latent design space generation. The encoder maps the design exemplar into a latent code. The latent code and the viewport image are fed into the generator. The generator then produces an image where the viewport matches the viewport image and the TF matches the design exemplar. During training, the encoder and the generator are trained together based on losses computed by the discriminator and pixel-wise comparison.

the latent code. Although a larger dimension could contain more information, overestimating it too much could lead to unnecessary weights and overfitting [1]. Based on our pilot experiment, we did not observe significant performance improvement on the “imitation task” as the dimension exceeds 64. Additionally, we found that the latent dimension is not a highly sensitive hyperparameter across the datasets we tested. The design exemplar is on a white or black background without the alpha channel. The encoder consists of a series of down-sampling blocks. Each down-sampling block has a convolution layer and a leaky ReLU activation layer (except for the last one). The convolution layers use small filters to extract local features from the previous layers, which are commonly used for feature extraction and representation learning tasks. The height and width of the feature maps will reduce by half after each convolution layer. The leaky ReLU layers introduce nonlinearity into the model. We choose the leaky ReLU activation function to mitigate the “dying ReLU” problem, which can occur when neurons in the network are stuck at zero activation and stop learning. Except for the first and last down-sampling blocks, we add an extra instance normalization layer [41] between the convolution layer and the leaky ReLU layer to stabilize training. We use instance normalization layer instead of batch normalization layer due to its ability to preserve the variations in style and content features within a single design exemplar.

### 4.3 Generator

The architecture of the generator is shown in Fig. 5 (b), which accepts a latent code and a viewport image as input and outputs a neural-rendered image. Different from the design exemplar, which only has RGB channels, the viewport image and the neural-rendered image have RGBA channels. We made this design choice to allow users the freedom to decide whether to display the neural-rendered image with a white background or a black background during latent design space exploration. We use a viewport image instead of raw viewport parameters to represent viewport information because previous works [1, 15] indicate that image input can provide the neural renderer with richer spatial information and increase the stability of training. The overall architecture of the generator is a U-net [37], consisting of a series of down-sampling blocks followed by a series of up-sampling blocks. Skip connections are added between each down- and up-sampling block of the same spatial shape. This helps to enforce spatial consistency in the neural-rendered image. The interior structure of the down-sampling blocks is similar to that of the down-sampling blocks of the encoder. In the up-sampling blocks, we use the inverse convolution layer instead of the convolution layer. Within every up-sampling block, a dropout layer is added after the instance normalization layer to avoid over-fitting, as suggested by [18].

### 4.4 Discriminator

The architecture of the discriminator is shown in Fig. 5 (c), which takes either the ground truth image  $I_{GT}$  or the neural-rendered image  $I_{pred}$  as the input and outputs the possibility that this image is a ground truth image. The discriminator consists of a series of down-sampling

blocks, which are also used in the encoder and the generator. In the last down-sampling block, we use the sigmoid activation layer instead of the leaky ReLU activation layer. This ensures the predicted probability to fall within the range of 0-1, where 1 represents that the discriminator regards the input as a ground truth image. Note that the output of the last sampling block still has a width of 32 and a height of 32 instead of a single scalar probability. This is because the discriminator is indeed  $32 \times 32$  small discriminators, each with a receptive field of  $32 \times 32$  in the input image. Previous work [18] has demonstrated that utilizing multiple small discriminators with small receptive fields, as opposed to a single large discriminator with a global receptive field, can enhance performance while maintaining the same number of parameters. We compute the final prediction of the discriminator by taking the average of the predicted probabilities from those small discriminators.

### 4.5 Losses

The loss  $L$  for the encoder and the generator is computed by combining the adversarial loss provided by the discriminator with the L1 (pixel-wise comparison) loss as follows:

$$L = \frac{1}{b} \sum_{i=0}^{b-1} (-\log D(I_{pred}^i) + \lambda_1 \|I_{pred}^i - I_{GT}^i\|_1) \quad (1)$$

where  $b$  is the batch size,  $D(I_{pred}^i)$  is the prediction result of the discriminator  $D$ ,  $\lambda_1$  is the weight of the L1 loss. Previous literature indicates that  $\lambda_1$  is relatively insensitive to set [1]. In practice, we find  $\lambda_1 = 1000$  stabilizes training without blurring the neural-rendered images.

The discriminator is also optimized by computing  $L_D$  during training, which is defined as follows:

$$L_D = -\frac{1}{b} \sum_{i=0}^{b-1} (\log D(I_{GT}^i) + \log(1 - D(I_{pred}^i))) \quad (2)$$

### 4.6 Training

This section describes how we prepare the training data and perform training.

**Training Data:** The set of design exemplars used as the training data implicitly define the target design space of transfer functions. Inspired by the transfer function sampling strategy by Berger et al. [1], we tailor a sampling strategy for design exemplar generation. The domain of the transfer function is first divided into several feature regions of interest. By doing this, uninteresting feature regions (e.g., regions of air or noise) can be removed from the target design space. Next, we assign a Gaussian transfer function (GTF) primitive to each feature region, which we refer to as the base GTF. For each round of sampling, we randomly choose several feature regions of interest, apply random shift to all of the parameters (color, opacity, mean, std) of the base GTFs in those regions, and combine them together to generate a final transfer function design. The parameters of the base GTF and the range for a random shift can be manually set to control the target design space. For example, we can bias the color (represented in HSL color space, independently sampled for each dimension) of features toward higher lightness and saturation. Once the transfer function  $T_{exp}$  for the design exemplar is generated, we render it with a random viewport  $V_{exp}$ . The viewport image is rendered with another random viewport  $V_{view}$  and the auxiliary transfer function  $T_{view}$ . The auxiliary transfer function is created by combining the assigned representative GTF for each feature region of interest. To create an informative viewport image, it is essential to adjust the std of each representative GTF such that the entire feature region is visible. Simultaneously, maintaining a low opacity for each representative GTF helps prevent issues with light/opacity saturation, thus preserving clarity in the image. The ground truth image is rendered with viewport  $V_{view}$  and transfer function  $T_{exp}$ . This sampling strategy can be easily extended to multi-dimensional transfer functions using the multi-dimensional GTF proposed by Kniss et al. [25]. In our experiment, we use the 2D transfer function to generate design space for tooth data since its features can be better separated with the 2D gradient intensity histogram.



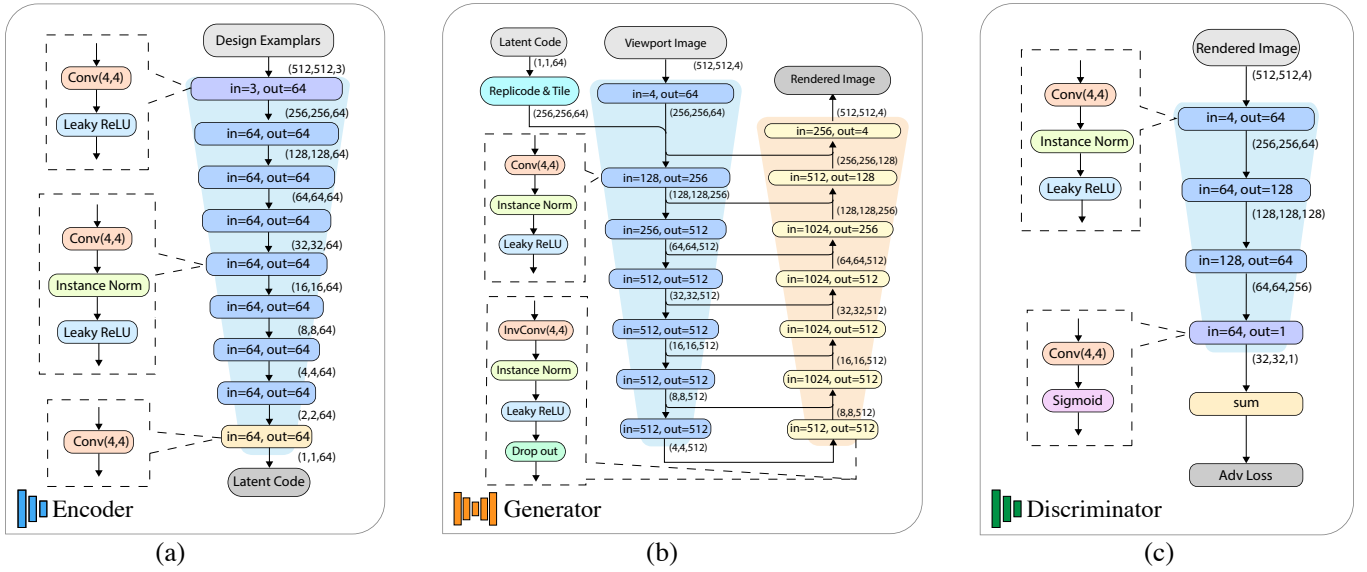


Fig. 5: Detail of the architecture of the encoder, the generator, and the discriminator. Blocks labeled with  $in=n, out=m$  are down-sampling blocks or up-sampling blocks with  $n$  input and  $m$  output channels. Other blocks are labeled with their operations.

**Training Process:** During training, the encoder, the generator, and the discriminator are optimized together. At the beginning, we initialize the weight of all networks using Xavier initialization [8]. For each iteration, we sample a batch of image triplets  $\{I_{exp}, I_{view}, I_{GT}\}$ . The encoder first takes the design exemplar  $I_{exp}$  as input and maps it into a latent code. The latent code and the viewport image are subsequently fed into the generator to obtain the predicted image  $I_{pred}$ . The weights of the discriminator are first updated according to the loss  $L_D$  defined in Eq. (2). Next, the weights of the encoder and the generator are updated according to the loss  $L$  defined in Eq. (1). We use Adam [23] as our weight optimizer. To stabilize the training, a small learning rate  $\alpha$  is preferred. We find that  $\alpha = 0.0002$  stabilizes the training in our case. We set the remaining hyperparameters of the optimizer to be consistent with previous works [6, 18] on image-to-image generative tasks ( $\beta_1 = 0.5, \beta_2 = 0.999$ ).

## 5 LATENT DESIGN SPACE EXPLORATION

To demonstrate the benefits of generating a latent design space, we propose a set of interactive tools to help users explore this latent design space. With these tools, users can conduct exploration without going through the process of using transfer function widgets.

### 5.1 Outline

To provide an overview of the design space, we perform Principal Component Analysis (PCA) [33] on the latent code of 10,000 randomly generated design exemplars to project them onto a 2D plane (see Fig. 1 b). Similar to the original design gallery method [31], the outline view arranges the design exemplars in a manner that allows users to observe and analyze the patterns that exist across the entire design space. For example, in Fig. 1 b, design exemplars with similar color/opacity of features are clustered together. The outline view also serves as a springboard for exploration, allowing users to select some design exemplars as their starting point. To prevent the thumbnails of the design exemplars from occluding each other, we display them hierarchically. The 2D plane supports zooming in and out. All the design exemplars are represented as tiny gray dots initially to provide a rough context for the global scope. We randomly sample a batch of design exemplars (the number is set to 100 during the experiments) to display their corresponding thumbnail images. If no thumbnails of interest are found, they can be resampled from the broader set. When a certain region of the projected plane is found to be interesting, users can drag a box over this region and click the box to focus on it. New thumbnails will be resampled among the gray dots in this local region. Users can click on a thumbnail to view its details in Fig. 1 e. When users drag on the image in Fig. 1 e to orbit around the volume, a

corresponding new viewport image will be rendered and subsequently enhanced by the generator. All the design exemplars that have been checked during exploration will be saved to the history view, allowing users to restore them at any point (Fig. 1 c).

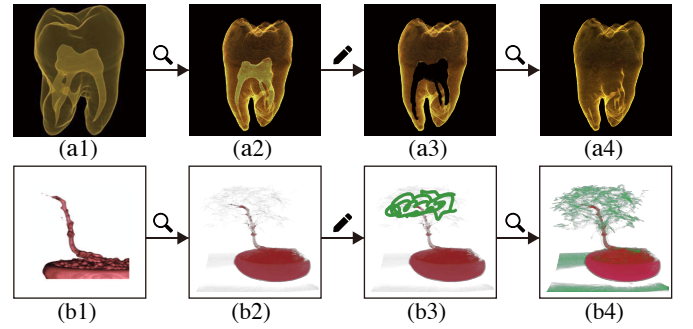


Fig. 6: The query operation. (a1, b1) are DVR images rendered by other researchers [25, 50]. Use (a1, b1) to query the latent design space can obtain (a2, b2). If we do some simple hand draws like (a3, b3) and use them to query the latent design space again, we can obtain (a4, b4).

### 5.2 Query

Although the outline view and hierarchical display strategy enable users to browse the designs they might want, sometimes users already have a target design in mind. In such cases, users want to quickly locate their target design in the design space or at least know if it contains any similar design as a good starting point. To address this, we provide a query tool. At any time during exploration, the user can click the  $\mathcal{Q}$  in Fig. 1 a to upload a query image. The encoder can analyze this query image and map it into a latent code in the latent design space. This latent code will be projected onto the outline view. The generator will take this latent code and render it as a new design exemplar.

Fig. 6 shows example images that can be used for the query operation. The query image can be a DVR image created by our DVR renderer (used for generating training set), or by any other DVR renderers. Moreover, we have found that the encoder can even understand hand-drawn modifications made to DVR images. Fig. 6 provides two illustrative examples: In (a1), we show a DVR image of the tooth dataset generated by Kniss et al. [25]. If this image is used as a query image, we obtain a similar new design exemplar (a2) within the latent design space. If we use an external drawing tool to cover the pulp with the background color and upload this image (a3) as a new query image, we obtain a design exemplar (a4) without pulp. In (b1), we show a DVR image of

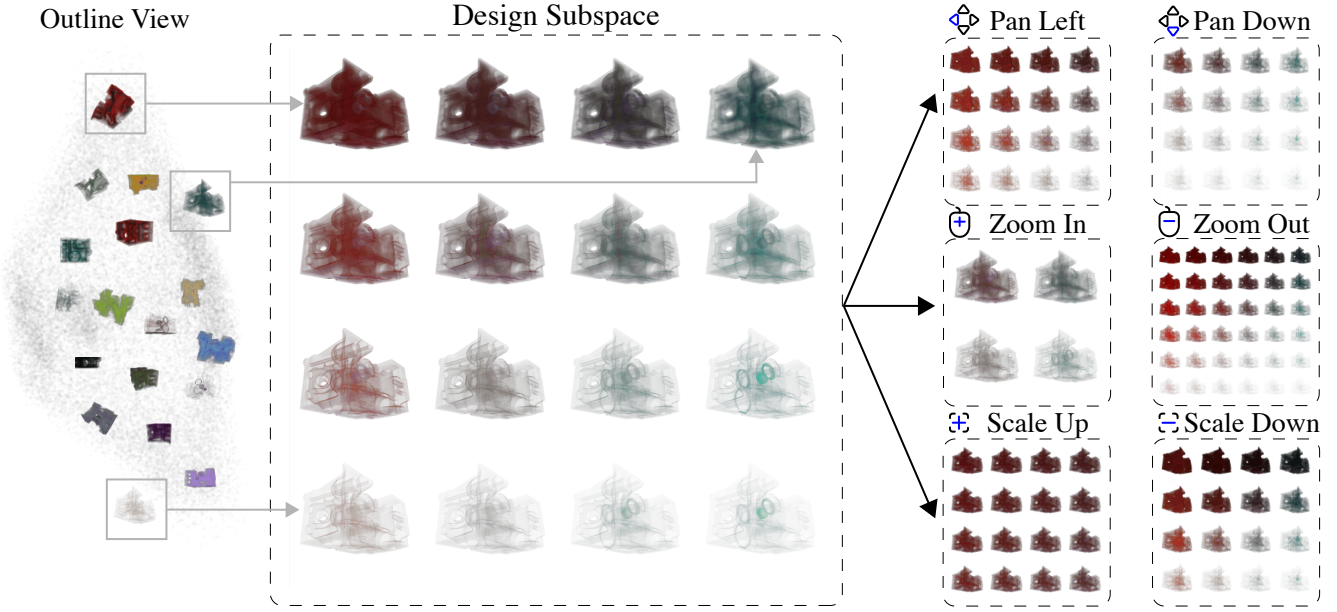


Fig. 7: The navigation operation. Some design exemplars are selected from the outline view to explore the design subspace among them. Intuitive navigation operations like panning, zooming, and scaling can be done to search for the target design in this design subspace.

the bonsai dataset generated by Wu and Qu [50]. If this image is used as a query image, we obtain a similar new design exemplar (b2) within the latent design space. If we roughly sketch some green leaves and use this image (b3) as a new query image, we obtain a design exemplar (b4) with the corresponding features displayed as green.

### 5.3 Navigation

It is essential to provide users with an effective tool for navigating the high-dimensional design space behind the discrete design exemplars. Although projecting those design exemplars onto a 2D plane and navigating this 2D plane provides a rough overview of the design space, a lot of information is distorted or lost during the projection process. To address this, we provide users with a navigation view (see Fig. 1, d). Users can freely select two or three design exemplars of interest from the outline view (Fig. 1 b) or history view (Fig. 1 c) to explore their design subspace in this navigation view. With two selected design exemplars, we can generate a 1D subspace that passes through their latent codes. With three selected design exemplars, we can generate a 2D subspace that passes through their latent codes. For example, if the user chooses three design exemplars from the outline view as shown in Fig. 7, a 4×4 array of neural-rendered images will be displayed to present the plane defined by the latent codes of these three design exemplars. The user can then click “←/→” to move towards another part of the plane, click “+/-” to adjust the difference between neighboring design exemplars in this plane, scroll the mouse to zoom in for detail comparison or zoom out for an overview. Users can freely select any design exemplar they like during navigation and add it to the history view for later exploration.

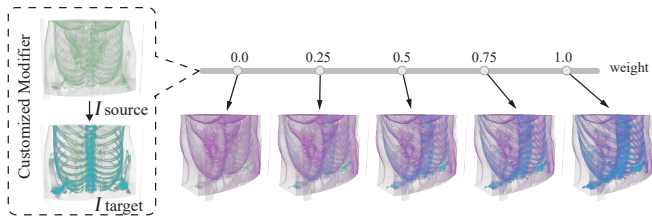


Fig. 8: The modification operation. The direction from the latent code of  $I_{source}$  to the latent code of  $I_{target}$  can be defined as a customized modifier of semantic direction “add blue skeleton”. This modifier can be applied to other designs to add a blue skeleton to them.

### 5.4 Modification

Users may be interested in a design exemplar and want to modify it. To address this, we designed the modification tool. The modification tool is designed based on the finding that many semantically meaningful directions can be discovered within the latent space of generative networks [39]. The user can choose one design exemplar  $I_{source}$  as the source and another design exemplar  $I_{target}$  as the target. The direction from the latent code of  $I_{source}$  to  $I_{target}$  can be defined as a customized modifier, denoted by  $T_{dir}$ . This  $T_{dir}$  can be used to modify another design exemplar with latent code  $T_{raw}$  as follows:

$$T_{new} = T_{raw} + \lambda_2 * T_{dir} \quad (3)$$

where  $\lambda_2$  is used to control the degree of modification.

Fig. 8 demonstrates an example of modification. The user first chooses a design exemplar without skeleton as  $I_{source}$  and another design exemplar with blue skeleton as  $I_{target}$  to define a customized modifier. This modifier follows the semantic direction “add blue skeleton”. If the user applies this modifier to another design exemplar, the skeleton of that design exemplar will become bluer and more visible. In practice, we find that the query and navigation tools can help users create customized modifiers with the semantics they want. For example, users can use the query result in Fig. 6 (a2) as  $I_{source}$ , and the result in Fig. 6 (a4) as  $I_{target}$  to create a modifier of semantic direction “remove pulp”. Users can also discover a helpful semantic direction during the exploration with the navigation view and save it as a modifier.

## 6 EXPLICIT TRANSFER FUNCTION RECONSTRUCTION

When users obtain the target design represented as a neural-rendered design exemplar during latent design space exploration, they can reconstruct it back to an explicit transfer function by clicking the  $\Delta$  button in Fig. 1 f. The transfer function under reconstruction and its rendering result will be displayed in real-time in Fig. 1 g. The reconstruction process is an optimization process with a differentiable DVR renderer.

Differentiable rendering is a technique that allows the optimization of scene parameters (e.g., geometry, materials) from reference images [20]. We adapt the approach proposed by Weiss and Westermann [48] to perform differentiable DVR rendering. In our scenario, the transfer function  $T$  is the scene parameter to be optimized, while the neural-rendered image  $I_{pred}$  serves as the reference image. The optimization objective is to minimize the pixel-wise difference between the rendering output and the reference image. The optimization process for each iteration can be divided into two main stages: the forward pass

	Dataset	Resolution	Precision	Size(MB)	Training Samples	Image SSIM	Image PSNR	Color EMD
(a)	Bonsai	256×256×256	byte	16.0	64,000	0.982	33.1	0.00518
	Engine	256×256×110	byte	6.87	64,000	0.976	32.5	0.00498
	CT-Chest	384×384×240	byte	33.7	64,000	0.990	36.7	0.00308
	Torso Phantom	256×256×256	byte	16.0	128,000	0.983	30.1	0.00472
	Tooth	256×256×256	byte	10.0	128,000	0.988	34.8	0.00297












































































	Bonsai			Engine			CT-Chest			Torso Phantom			Tooth		
	$I_{exp}$	$I_{pred}$	$I_{GT}$	$I_{exp}$	$I_{pred}$	$I_{GT}$	$I_{exp}$	$I_{pred}$	$I_{GT}$	$I_{exp}$	$I_{pred}$	$I_{GT}$	$I_{exp}$	$I_{pred}$	$I_{GT}$
(b)															
															
															
															
															

Fig. 9: In (a), we provide quantitative results of the “imitation task” across different volumetric data. In (b), we show qualitative results.  $I_{exp}$  denotes the design exemplar,  $I_{pred}$  denotes the neural-rendered result,  $I_{GT}$  denotes the ground truth. The last two rows show typical artifacts (e.g., incorrect color mapping, wrong details) for each dataset.

renders an image with the current transfer function  $T$  like a traditional ray-casting-based DVR renderer, and computes the absolute difference between the rendered image and the reference image as the loss function; the backward pass computes the gradient of the transfer function  $T$  with respect to the loss function using back propagation, and then updates  $T$  using gradient descent.

The reconstructed transfer function could contain high-frequency variations that are less important for humans. Therefore, a smoothing prior term is added to the loss term to encourage the reconstructed transfer function to be smooth. In practice, we set the weight of the smoothing prior term to be 0.4, following Weiss and Westerman [48].

When the optimization process converges, the transfer function  $T$  can be used to obtain a DVR rendering result that is similar to the given neural-rendered image. Since the reconstructed transfer function is texture-based, it can be exported and used in other DVR renderers.

## 7 RESULTS

### 7.1 Implementation Details

Our experiments used the following volume datasets: the Bonsai<sup>1</sup>, the Engine, the CT Chest<sup>1</sup>, the Torso Phantom, and the Tooth. The DVR renderer was adapted from the CUDA implementation of Weiss [48], which is based on the basic emission-absorption model [32]. This DVR renderer can switch between two modes: (1) in non-differentiated mode, the renderer just performs like a traditional DVR renderer. (2) in differentiated mode, the renderer becomes a differentiable DVR renderer. Since the tooth dataset is better classified with 2D transfer functions, we extended Weiss’s renderer such that it can support 2-dimensional transfer functions in both non-differentiated mode and differentiated mode. We used this extended renderer in non-differentiated mode to generate all the images for the neural network training. We used this extended renderer in differentiated mode as the differentiable DVR renderer for transfer function reconstruction. The encoder and the generator were implemented in PyTorch and trained on a symmetric multiprocessing node with 8 NVIDIA 3090 GPUs. The training time ranged from 10 to 30 hours, depending on the complexity of the dataset and the transfer function design space. After training, the encoder

took an average of 2.19 ms for inference, while the generator took an average of 4.56 ms for inference. All the images generated during our experiments were of 512×512 resolution. The interactive applications were implemented based on a web server/client framework. The interface was implemented with D3.js on the client side. All the rendering, network inference, and algorithm execution are done on the server side.

### 7.2 Evaluation for Latent Design Space Generation

As described in Sec. 4, we use an “imitation task” as the auxiliary task to train the encoder and the generator for latent design space generation. For each volume dataset, we evaluated the performance of this “imitation task” on a separate set of 1,000 volume-rendered images, which are not seen during the training phase. We used peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM), and color histogram’s earth mover’s distance (Color EMD) as our evaluation metrics. These three indicators complement each other: PSNR calculates the average mean squared error between the pixels of two images. SSIM takes into account the structural information of images, such as edges, textures, and contrast. Color EMD measures the similarity of color distribution between two images.

In Fig. 9 (a), we report the evaluation results of average PSNR, SSIM, and EMD across all datasets. Those metrics indicate that overall our encoder can successfully analyze the given design exemplar and find a latent code that generates an image resembling the ground truth. The qualitative experimental results in Fig. 9 (b) also confirm this.

To investigate how the color space, the viewpoint image, and the viewpoint of the design exemplar impact the generation result, we conducted comparative experiments for each of these factors.

As shown in Fig. 10 (a), we compared two restricted color spaces for the CT-Chest dataset. Color space B (H = 0-360, S = 0.8-1.0, L = 0.4-0.6) is larger than color space A (H = 0-90, S = 0.8-1.0, L = 0.4-0.6). Given the same training set size, sampling within color space A can always reach better generation quality. This result indicates that we can use fewer training samples and iterations to reach satisfactory generation quality by limiting the color space.

In Fig. 10 (b), we compared the generation results among different viewpoint images for the CT-Chest dataset. As we discussed in Sec. 4.6, to preserve details, the opacity of the transfer function  $T_{view}$  should be low enough to avoid opacity/light saturation.  $T_{view1}$  is a transfer

<sup>1</sup><https://klacansky.com/open-scivis-datasets>



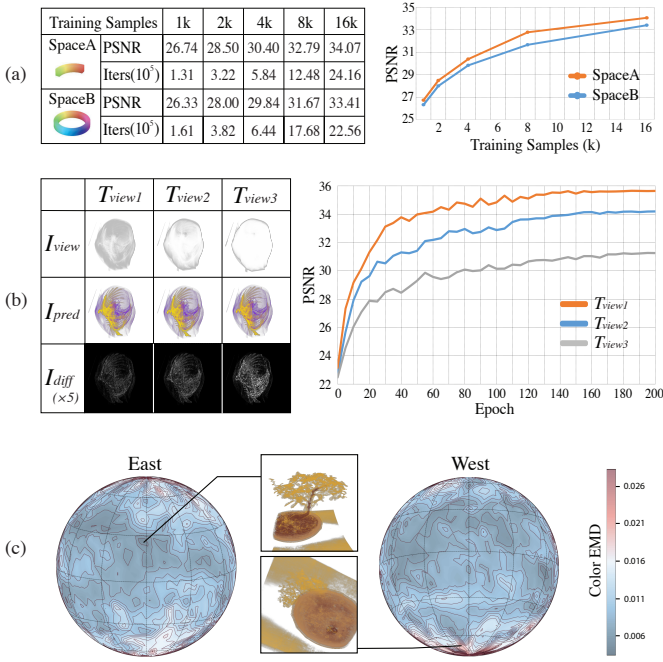


Fig. 10: Experiments on different factors that impact the generation result. In (a), we compare two color spaces for the CT-Chest dataset. Given the same training set size, sampling within a smaller color space can achieve a better generation result. In (b), we compare different viewport images for the CT-Chest dataset. A non-saturated viewport image preserves more details and leads to better generation result. In (c), we compare different viewpoints of the design exemplar for the bonsai dataset. Viewports from which most features are clearly presented lead to better generation results.

function that creates a viewport image without saturated pixels.  $T_{view2}$  is a transfer function that leads to some saturated pixels in the viewport image.  $T_{view3}$  is a transfer function that leads to a lot of saturated pixels in the viewport image. The only difference between  $T_{view1}$ ,  $T_{view2}$ , and  $T_{view3}$  is the maximum opacity for each GTF. The result shows that  $T_{view1}$  can reach a better generation result than  $T_{view2}$  and  $T_{view3}$ .

In Fig. 10 (c), we conducted an experiment to show how the fidelity of the generated result changes for different viewpoints on a sphere. The spatial pattern illustrates that the networks tend to favor viewpoints where most features are clearly presented, which resembles human preference. In practice, we could forego sampling for viewpoints not of interest to help the network concentrate on the rest of the viewpoints, which ultimately depends on the user’s needs.

### 7.3 Experiments for Latent Design Space Exploration

We demonstrate how users can explore the latent design space with the interactive tools through an example case study with the Torso Phantom dataset. We recommend readers watch our associated video to see more exploration results across different datasets.

Fig. 11 illustrates how to find a target transfer function design in the latent design space in a few steps. The user first chooses a design exemplar **a** in the outline view as a starting point. However, this design exemplar contains some features not of interest that the user wants to subjugate, so the user manually creates simple hand drawings to cover those features with the background color (see **b**). **b** is then uploaded as a query image to obtain a design exemplar without those features (see **c**). However, the user is still not satisfied with the color scheme of the current visible features. Therefore, the user once again uses the digital drawing pen to recolor some features (see **d**) and upload it to query a design exemplar with similar colors (see **e**). Nevertheless, the user finds that **e** still contains some features not of interest. To deal with this, the user defines the direction from the latent code of **a** to the latent code of **c** as a customized modifier of semantic direction

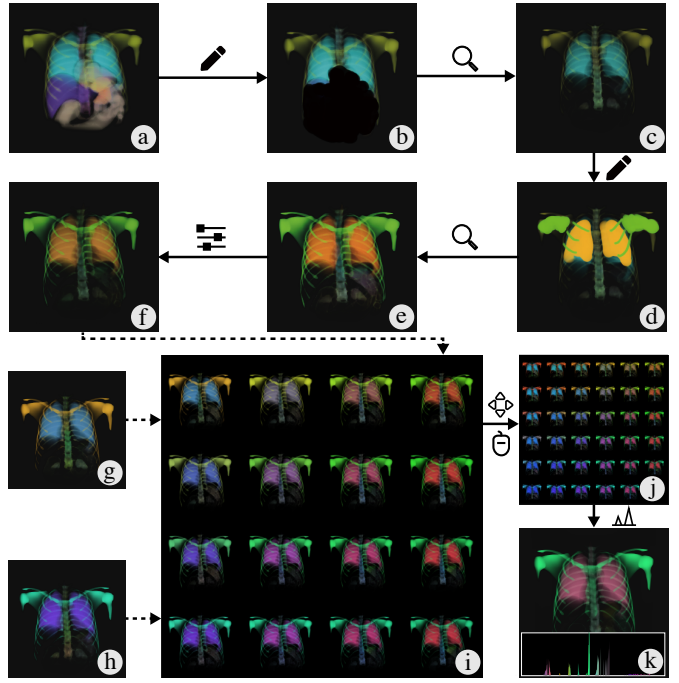


Fig. 11: Latent design space exploration with the Torso Phantom dataset. Image-based exploration tools are utilized interchangeably to reach the final transfer function design. No operation in TF domain is required.

“subjugate those uninteresting features”, and applies it to **e** to obtain **f**. Taking similar operations, the user creates another two candidate design exemplars (see **g** and **h**). To make a final decision, the user puts **f**, **g** and **h** into the navigation view to create a design subspace (see **i**) among them. Subsequently, the user browses through this design subspace using our interactive navigation operations to decide the final design (see **j**). Finally, the user clicks the  $\Delta\Delta$  button to reconstruct the explicit transfer function from the user’s design. The reconstructed transfer function and its rendering result is shown in **k**. All the operations during exploration are image-based and easy to grasp. The user is not required to do any operations in the TF domain.

Our interface provides a flexible framework under which users can customize their own workflow to explore the transfer function design space. We have found that some of our early users have discovered their own creative ways of using our interface. For example, one could use an external image editing tool to blend two DVR images and upload the blended result as a query image to achieve DVR image blending, similar to Wu et al. [50]. Users might utilize our modification tool to create interpolated results to achieve transfer function animations [29] [49]. Moreover, users can continuously discover and gather customized modifiers during the exploration process. However, when the user already has a specific modification target for a specific transfer function, they are more likely to find a desired tool from an interface that provides a rich set of tools with explicit modification purposes (e.g., the WYSI-WYG editor by Guo et al. [11]). Compared with those tools that allow only one modification at a time and rely on initial transfer functions, our interface excels in aiding users in efficiently navigating a vast design space to find the desired transfer function. In the future, we will investigate how to effectively integrate more image-based modification tools to further improve the usability of our interface.

### 7.4 Experiments for Transfer Functions Reconstruction

Weiss and Westermann [48] have demonstrated that the differentiable DVR renderer is capable of effectively performing the task of transfer function reconstruction. In their experiment, a traditional renderer is first used to render a reference DVR image with a manually set transfer function. Then, the differentiable DVR renderer is initialized with a random transfer function and a viewport the same as the reference

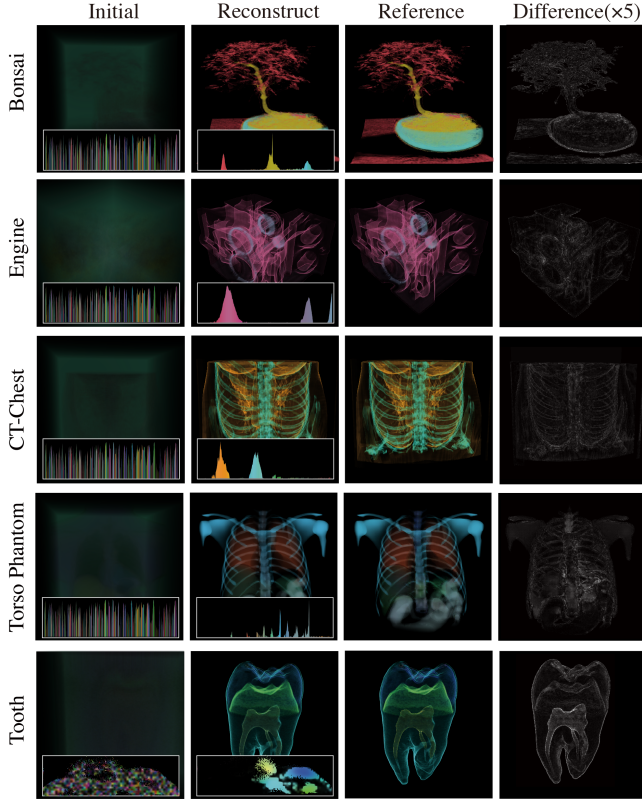


Fig. 12: TF reconstruction results across different datasets. From left to right: initial transfer function and rendering result; reconstructed transfer function and rendering result; the reference image; the pixel difference (x5 for better perception).

image. The pixel-wise difference between the reference image and the image rendered by the differentiable DVR renderer is used as the loss function to optimize the transfer function of the differentiable DVR renderer. Once convergence is achieved, the image rendered by the differentiable DVR renderer should be similar to the reference image.

The procedure of our reconstruction experiment is similar to that of Weiss and Westermann. The only distinction is that the reference image is now generated by our generator. Fig. 12 demonstrates the reconstruction results across different datasets. The reconstruction results closely match the references. An interesting observation is that the generator and the differentiable DVR renderer can complement each other: The rendering result of the generator provides a rough design direction of the transfer function but may introduce unreliable artifacts in details; The differentiable DVR renderer can optimize its transfer function toward the rough design direction provided by the neural-rendered result, while eliminating the artifacts due to its physically-based rendering process.

## 8 LIMITATION, DISCUSSION, AND FUTURE WORK

In this paper, we present a promising research direction of combining generative networks for design space exploration and differentiable DVR renderers for faithful visualization generation. We identify several potential avenues for future research to enhance our approach.

**Generation fidelity:** One limitation of our approach is that the fidelity of the generated images could decrease as the complexity of the

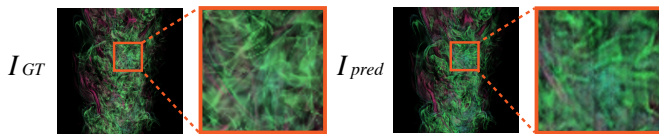


Fig. 13: The generated result on real-world combustion simulation data. When generating design exemplars for volumes with many high-frequency features, the details tend to be blurred and distorted.

volume increases. For example, when generating design exemplars with many high-frequency features, the details tend to be blurred/distorted (see Fig. 13 as an example). When there are many features clustering together within a certain area of the volume, they may occlude each other, which makes it harder for the neural network to identify them. As a result, the generated image might assign those features the wrong colors (see Fig. 9 (b), the lower part of the anatomic structures in the Torso Phantom dataset), and the accuracy of the “query” operation will also decrease. Although the current architecture setting works for most volume data we tested, it might need to be changed for more complex real-world datasets. In the future, we will experiment with more architectures for the encoder and the generator.

**Training data:** The generation of training data could also be a research topic on its own. Generating a representative dataset for all possible transfer functions (TFs) is challenging and expensive due to the vastness of the TF space. Currently, we try to use Gaussian primitives to generate representative TFs, following the approach of Berger et al. [1]. Although Gaussian primitives can be used to achieve transfer functions of various shapes, they still contain bias. For example, the user may want to use an alternative primitive with a sharp ramp to show individual parts of the data opaquely, whereas the Gaussian primitives tend to exhibit a bias towards smoother ramps. To support customizing TF primitives and bias toward certain colors and opacities, more interactive tools can be developed to help efficiently inject user preference into the dataset generation process.

**Training time:** At present, the training time ranges from 10 to 30 hours, depending on the complexity of the dataset and the transfer function design space. We believe that there is potential to further reduce the training time via transfer learning. The same type of volumetric data usually shares similar feature shapes and scalar value distributions. Therefore, we can pre-train a backbone model, capturing common feature representations for a class of volumetric data. By fine-tuning this model on new volumetric data instead of training a model from scratch, we may further reduce the training time.

**Latent space exploration:** Further research can be conducted to enhance latent space exploration by designing additional forms of latent space and expanding the collection of interactive exploration tools. The latent space generated by our current architecture should be just “a” latent space among all the possible latent spaces. For instance, we can benefit from learning another latent space where there are both continuous and discrete dimensions [6]. Furthermore, we can design different architectures to better match the “Platonic latent space” for different classes of volume datasets and various exploration scenarios. Moreover, we intend to incorporate more interactive tools to facilitate the exploration process. For example, adding a “filtering” tool (based on image processing or latent code classification) that allows users to filter out some colors/features from the 2D embedding view could be helpful.

## 9 CONCLUSION

In this paper, we present Differentiable Design Galleries, an image-based transfer function design approach based on deep learning and differentiable rendering. We tailor a neural rendering task to learn a latent design space, which represents various types of implicit transfer functions in a continuous manifold. Users can explore this latent design space intuitively with the set of interactive tools we provide. Furthermore, our approach can help users reconstruct their target design back to the explicit transfer function with the aid of a differentiable direct volume renderer. The effectiveness of our approach is demonstrated through experiments conducted on real volumetric data. We believe our approach has the potential to be extended to more complex volumetric data and more advanced exploration methods.

## ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. U22B2034) and Fundamental Research Funds for the Central Universities (226-2022-00235). This work is also supported by Natural Science Foundation of Zhejiang Province, China (No. LY21F020029). The authors would like to thank the anonymous



reviewers for their valuable comments. The authors wish to thank Sebastian Weiss from the Technical University of Munich for his help on differentiable DVR. The authors wish to thank Can Liu from Peking University for his help on neural rendering. Finally, the authors would like to thank Oliver Woodman for proofreading earlier drafts.

## REFERENCES

- [1] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 25(4):1636–1650, 2019. doi: [10.1109/TVCG.2018.2816059](#) 2, 3, 4, 9
- [2] C. Correa and K.-L. Ma. Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1380–1387, 2008. doi: [10.1109/TVCG.2008.162](#) 2
- [3] C. Correa and K.-L. Ma. The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1465–1472, 2009. doi: [10.1109/TVCG.2009.189](#) 2
- [4] C. Correa and K.-L. Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):192–204, 2011. doi: [10.1109/TVCG.2010.35](#) 2
- [5] F. de Moura Pinto and C. M. Freitas. Design of multi-dimensional transfer functions using dimensional reduction. In *Proceedings of the 9th Joint Eurographics/IEEE VGTC conference on Visualization*, pp. 131–138. IEEE, Piscataway, 2007. doi: [10.2312/VisSym/EuroVis07/131-138](#) 2
- [6] E. Dupont. Learning disentangled joint continuous and discrete representations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, p. 708–718. Curran Associates Inc., New York, 2018. doi: [10.48550/arXiv.1804.00104](#) 5, 9
- [7] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. A K Peters/CRC Press, New York, 2006. doi: [10.1201/b10629](#) 1
- [8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256. PMLR, Sardinia, 2010. 5
- [9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *CoRR*, 2014. doi: [10.48550/arXiv.1406.2661](#) 3
- [10] H. Guo, W. Li, and X. Yuan. Transfer function map. In *Proceedings of the IEEE Pacific Visualization Symposium*, pp. 262–266. IEEE, Piscataway, 2014. doi: [10.1109/PacificVis.2014.24](#) 2
- [11] H. Guo, N. Mao, and X. Yuan. Wysiwyg (what you see is what you get) volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2106–2114, 2011. doi: [10.1109/TVCG.2011.261](#) 2, 8
- [12] M. Haidacher, S. Bruckner, A. Kanitsar, and M. E. Gröller. Information-based transfer functions for multimodal visualization. In *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*, pp. 101–108. Wiley, New Jersey, 2008. doi: [10.2312/VCBM/VCBM08/101-108](#) 2
- [13] T. He, L. Hong, A. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In *Proceedings of the IEEE Visualization*, pp. 227–234. IEEE, Piscataway, 1996. doi: [10.1109/VISUAL.1996.568113](#) 2
- [14] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. G. Nashed, and T. Peterka. Insitunet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):23–33, 2020. doi: [10.1109/TVCG.2019.2934312](#) 2, 3
- [15] F. Hong, C. Liu, and X. Yuan. Dnn-volvis: Interactive volume visualization supported by deep neural network. In *Proceedings of the IEEE Pacific Visualization Symposium*, pp. 282–291. IEEE, Piscataway, 2019. doi: [10.1109/PacificVis.2019.00041](#) 2, 3, 4
- [16] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991. doi: [10.1016/0893-6080\(91\)90009-T](#) 3
- [17] C. Y. Ip, A. Varshney, and J. JaJa. Hierarchical exploration of volumes using multilevel segmentation of the intensity-gradient histograms. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2355–2363, 2012. doi: [10.1109/TVCG.2012.231](#) 2
- [18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 5967–5976. IEEE, Piscataway, 2017. doi: [10.1109/CVPR.2017.632](#) 3, 4, 5
- [19] D. Jönsson, M. Falk, and A. Ynnerman. Intuitive exploration of volumetric data using dynamic galleries. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):896–905, 2016. doi: [10.1109/TVCG.2015.2467294](#) 2
- [20] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. Differentiable rendering: A survey. *CoRR*, 2020. doi: [10.48550/arXiv.2006.12057](#) 6
- [21] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of IEEE Symposium on Volume Visualization*, pp. 79–86. IEEE, Piscataway, 1998. doi: [10.1109/SVV.1998.729588](#) 2
- [22] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Moller. Curvature-based transfer functions for direct volume rendering: methods and applications. In *Proceedings of the IEEE Visualization*, pp. 513–520. IEEE, Piscataway, 2003. doi: [10.1109/VISUAL.2003.1250414](#) 2
- [23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, 2014. doi: [10.48550/arXiv.1412.6980](#) 5
- [24] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002. doi: [10.1109/TVCG.2002.1021579](#) 2
- [25] J. Kniss, S. premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *Proceedings of the IEEE Visualization*, pp. 497–504, 2003. doi: [10.1109/VISUAL.2003.1250412](#) 4, 5
- [26] J. Kniss, R. Van Uiter, A. Stephens, G.-S. Li, T. Tasdizen, and C. Hansen. Statistically quantitative volume visualization. In *Proceedings of the IEEE Visualization*, pp. 287–294. IEEE, Piscataway, 2005. doi: [10.1109/VISUAL.2005.1532807](#) 2
- [27] P. Ljung, J. Krüger, E. Groller, M. Hadwiger, C. D. Hansen, and A. Ynnerman. State of the art in transfer functions for direct volume rendering. In *Computer Graphics Forum*, vol. 35, pp. 669–691. Wiley-Blackwell, New Jersey, 2016. doi: [10.1111/cgf.12934](#) 1, 2
- [28] C. Lundstrom, P. Ljung, and A. Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579, 2006. doi: [10.1109/TVCG.2006.100](#) 2
- [29] K.-L. Ma. Image graphs-a novel approach to visual data exploration. In *Proceedings of the IEEE Visualization*, pp. 81–88. IEEE, Piscataway, 1999. doi: [10.1109/VISUAL.1999.809871](#) 8
- [30] R. Maciejewski, I. Woo, W. Chen, and D. Ebert. Structuring feature space: A non-parametric method for volumetric transfer function generation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1473–1480, 2009. doi: [10.1109/TVCG.2009.185](#) 2
- [31] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 389–400. ACM, New York, 1997. doi: [10.1145/258734.258887](#) 2, 5
- [32] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. doi: [10.1109/2945.468400](#) 7
- [33] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901. doi: [10.1080/14786440109462720](#) 5
- [34] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Raghu, R. Machiraju, and J. Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001. doi: [10.1109/38.920623](#) 1, 2
- [35] F. d. M. Pinto and C. M. Freitas. Volume visualization and exploration through flexible transfer function design. *Computers & Graphics*, 32(4):420–429, 2008. doi: [10.1016/j.cag.2008.04.004](#) 2
- [36] J. Prauchner, C. Freitas, and J. Comba. Two-level interaction approach for transfer function specification. In *Brazilian Symposium on Computer Graphics and Image Processing*, pp. 265–272. IEEE, Piscataway, 2005. doi: [10.1109/SIBGRAPI.2005.52](#) 2
- [37] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241. Springer, Cham, 2015. doi: [10.1007/978-3-319-24574-4\\_28](#) 4



- [38] C. R. Salama, M. Keller, and P. Kohlmann. High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1021–1028, 2006. doi: [10.1109/TVCG.2006.148](#) 2
- [39] Y. Shen and B. Zhou. Closed-form factorization of latent semantics in gans. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 1532–1540. IEEE, Piscataway, 2021. doi: [10.1109/CVPR46437.2021.00158](#) 6
- [40] F.-Y. Tzeng and K.-L. Ma. A cluster-space visual interface for arbitrary dimensional classification of volume data. vol. 2004, pp. 17–24, 338. Wiley, New Jersey, Jan 2004. doi: [10.2312/VisSym/VisSym04/017-024](#) 2
- [41] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, 2016. doi: [10.48550/arXiv.1607.08022](#) 4
- [42] C. Wang and J. Han. D14scivis: A state-of-the-art survey on deep learning for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 29(8):3714–3733, 2023. doi: [10.1109/TVCG.2022.3167896](#) 2
- [43] Y. Wang, W. Chen, J. Zhang, T. Dong, G. Shan, and X. Chi. Efficient volume exploration using the gaussian mixture model. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1560–1573, 2011. doi: [10.1109/TVCG.2011.97](#) 2
- [44] Y. Wang, J. Zhang, D. J. Lehmann, H. Theisel, and X. Chi. Automating transfer function design with valley cell-based clustering of 2d density plots. In *Computer Graphics Forum*, vol. 31, pp. 1295–1304. Wiley-Blackwell, New Jersey, 2012. doi: [10.1111/j.1467-8659.2012.03122.x](#) 2
- [45] J. Weiss and N. Navab. Deep direct volume rendering: Learning visual feature mappings from exemplary images. *CoRR*, 2021. doi: [10.48550/arXiv.2106.05429](#) 3
- [46] S. Weiss, M. Chu, N. Thuerey, and R. Westermann. Volumetric isosurface rendering with deep learning-based super-resolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(6):3064–3078, 2021. doi: [10.1109/TVCG.2019.2956697](#) 3
- [47] S. Weiss, M. Işık, J. Thies, and R. Westermann. Learning adaptive sampling and reconstruction for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2654–2667, 2022. doi: [10.1109/TVCG.2020.3039340](#) 3
- [48] S. Weiss and R. Westermann. Differentiable direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):562–572, 2022. doi: [10.1109/TVCG.2021.3114769](#) 3, 6, 7, 8
- [49] H.-C. Wong, U.-H. Wong, and Z. Tang. Direct volume rendering by transfer function morphing. In *Proceedings of the International Conference on Information, Communications and Signal Processing*, pp. 1–4. IEEE, Piscataway, 2009. doi: [10.1109/ICICS.2009.5397587](#) 8
- [50] Y. Wu and H. Qu. Interactive transfer function design based on editing direct volume rendered images. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1027–1040, 2007. doi: [10.1109/TVCG.2007.1051](#) 2, 5, 6, 8