Differentiable Direct Volume Rendering

Sebastian Weiss* and Rüdiger Westermann†

Technical University of Munich

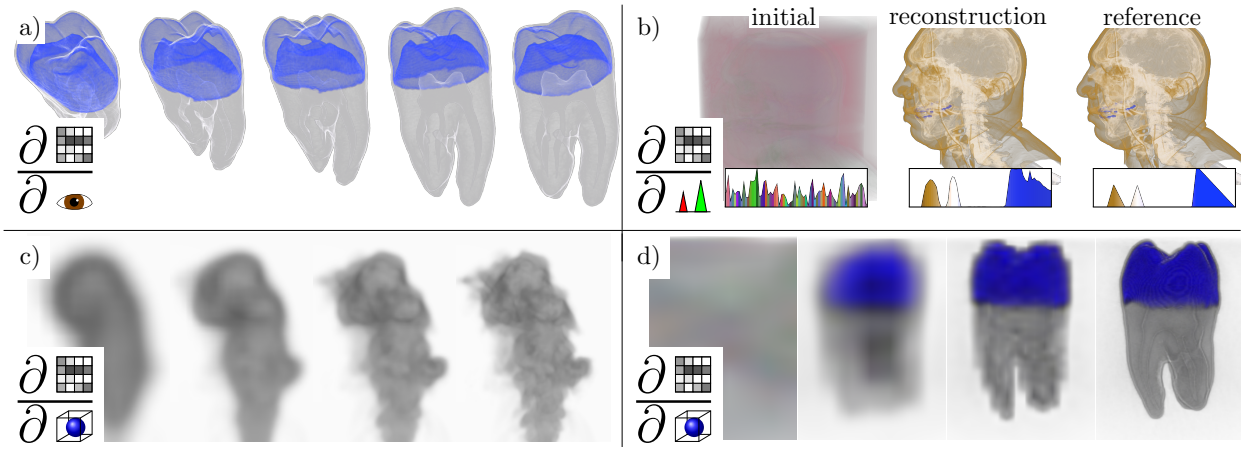

Fig. 1. A fully differentiable direct volume renderer is used for a) viewpoint optimization, b) transfer function optimization, and optimization of voxel properties using c) an absorption-only model and d) an emission-absorption model with rgbα transfer functions. a), c) and d) show intermediate results of the optimization process until convergence.

Abstract— We present a differentiable volume rendering solution that provides differentiability of all continuous parameters of the volume rendering process. This differentiable renderer is used to steer the parameters towards a setting with an optimal solution of a problem-specific objective function. We have tailored the approach to volume rendering by enforcing a constant memory footprint via analytic inversion of the blending functions. This makes it independent of the number of sampling steps through the volume and facilitates the consideration of small-scale changes. The approach forms the basis for automatic optimizations regarding external parameters of the rendering process and the volumetric density field itself. We demonstrate its use for automatic viewpoint selection using differentiable entropy as objective, and for optimizing a transfer function from rendered images of a given volume. Optimization of per-voxel densities is addressed in two different ways: First, we mimic inverse tomography and optimize a 3D density field from images using an absorption model. This simplification enables comparisons with algebraic reconstruction techniques and state-of-the-art differentiable path tracers. Second, we introduce a novel approach for tomographic reconstruction from images using an emission-absorption model with post-shading via an arbitrary transfer function.

Index Terms—Differentiable rendering, Direct Volume Rendering, Automatic Differentiation

1 Introduction

Differentiable direct volume rendering (DiffDVR) can serve as a basis for a multitude of automatic optimizations regarding external parameters of the rendering process such as the camera, the transfer function (TF), and the integration stepsize, as well as the volumetric scalar field itself. DiffDVR computes derivatives of the rendered pixel values with respect to these parameters and uses these derivatives to steer the parameters towards an optimal solution of a problem-specific objective (or loss) function. DiffDVR is in particular required when using neural network-based learning tasks, where derivatives need to be propagated seamlessly through the network for training end-to-end regarding the loss function.

While a number of approaches have been proposed for differentiable surface rendering [20], approaches focusing on differentiable rendering in the context of volume visualization are rare. For surface rendering, one objective is on the optimization of scene parameters like material properties, lighting conditions, or even geometric shape, to achieve matchings of synthetic and real images in computer vision tasks. Others have used implicit surface representations encoded via volumetric signed distance functions to derive analytic gradients for image-based shape reconstruction tasks [25, 35, 44]. These approaches assume opaque surfaces so that in each optimization iteration the gradient descent is with respect to the encoding of a single fragment per pixel. This is different from direct volume rendering applications, where the optimization needs to consider the contributions of many samples to a pixel color. This requires considering a large number of partial derivatives of pixel colors with respect to parameter or material changes, and to propagate them back into the volumetric field.

The differentiable rendering framework Mitsuba 2 [37] also provides a solution for direct volume rendering through Monte Carlo path tracing. However, Mitsuba directly applies so-called reverse-mode differentiation, which requires all intermediate derivatives to be saved for backpropagation. Thus, the memory required in direct volume rendering applications quickly exceeds the available system memory. This limits the approach to small volumetric grids and a small number of volume interactions that cannot faithfully optimize for small-scale structures. A follow-up work [36] addresses this issue but limits the differentiability to volume densities and colors.

*e-mail: sebastian13.weiss@tum.de
†e-mail: westermann@tum.de

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.xxxx/TVCG.201x.xxxxxxx

1.1 Contribution

This work presents a general solution for DiffDVR: differentiable Direct Volume Rendering using the emission-absorption model without multiple scattering. This requires analyzing approaches for automatic differentiation (AD) with respect to the specific requirements in direct volume rendering (DVR). So-called forward-mode approaches are efficient if the number of parameters is low, yet they become computationally too expensive with an increasing number of parameters, i.e., when optimizing for per-voxel densities in a volumetric field. The so-called reverse mode or adjoint mode records the operations and intermediate results in a graph structure. This structure is then traversed in reverse order during the backward pass that propagates the changes to the sample locations. However, this requires storing $O(kn)$ intermediate results, where n is the number of pixels and k the number of sample locations, and reversing the order of operations.

We show that a-priori knowledge about the operations performed in DVR can be exploited to avoid recording the operations in reverse-mode AD. We propose a custom computation kernel that inverts the order of operations in turn and derives the gradients used by AD. We further present a method for recomputing intermediate results via an analytic inversion of the light accumulation along the view rays. By this, intermediate results do not need to be recorded and the memory consumption of reverse-mode AD becomes proportional to $O(n)$.

As our second contribution, we discuss a number of use cases in which AD is applied in volume rendering applications (Fig. 1). These use cases demonstrate the automatic optimization of external parameters of the rendering process, i.e., the camera and the TF. Here the 3D density field is not changed, but the optimization searches for the external parameters that—when used to render this field—yield an optimal solution of a problem-specific loss function. In addition, we cover problems where the optimization is with respect to the densities. I.e., the field values are optimized so that an image-based loss function—after rendering the optimized field—yields an optimal solution. We consider inverse tomography by restricting the rendering process to an absorption-only model without a TF and optimize the densities using given images of the field. For this case, we compare our method against algebraic reconstruction techniques [49, 50] and Mitsuba 2 [36, 37]. Beyond that, and for the first time to our best knowledge, we show how to incorporate TFs and an emission-absorption model into tomographic reconstruction and deal with the resulting non-convex optimization problem.

DiffDVR is written in C++ and CUDA, and it provides seamless interoperability with PyTorch for a simple embedding into existing training environments with complex, potentially network-based loss functions. The code is made publicly available under a BSD license[1].

2 Related Work

Differentiable Rendering A number of differentiable renderers have been introduced for estimating scene parameters or even geometry from reference images, for example, under the assumption of local illumination and smooth shading variations [21, 26, 41, 42], or via edge sampling to cope with discontinuities at visibility boundaries [24]. Scattering parameters of homogeneous volumes have been estimated from observed scattering pattern [12]. Recently, Nimier-David *et al.* proposed Mitsuba 2 [37], a fully-differentiable physically-based Monte-Carlo renderer. Mitsuba 2 also handles volumetric objects, yet it requires storing intermediate results during the ray sampling process at each sampling point. This quickly exceeds the available memory and makes the approach unfeasible for direct volume rendering applications. Later, the authors have shown how to avoid storing the intermediate results [36], by restricting the parameters that can be derived to, e.g., only shading and emission. However, these methods are tailored for path tracing with multiple scattering and rely on Monte-Carlo integration with delta tracking. This makes them prone to noise and leads to long computation times compared to classical DVR methods without scattering. Our method, in contrast, does not require storing intermediate results and can, thus, use large volumes with arbitrary many sampling steps without resorting to a restricted parameter set. Furthermore, it does not impose restrictions on the parameters of the volume rendering process that can be differentiated.

Parameter Optimization for Volume Visualization An interesting problem in volume visualization is the automatic optimization of visualization parameters like the viewpoint, the TF, or the sampling stepsize that is required to convey the relevant information in the most efficient way. This requires at first hand an image-based loss function that can be used to steer the optimizer toward an optimal parameter setting. To measure a viewpoint's quality from a rendered image, loss functions based on image entropy [7, 19, 46, 51] or image similarity [47, 57] have been used. For volume visualization, the relationships between image entropy and voxel significance [5] as well as importance measures of specific features like isosurfaces [45] have been considered. None of these methods, however, considers the rendering process in the optimization process. Instead, views are first generated from many viewpoints, e.g., by sampling via the Fibonacci sphere algorithm [28], and then the best view regarding the used loss function is determined. We envision that by considering the volume rendering process in the optimization, more accurate and faster reconstructions can be achieved.

Another challenging problem is the automatic selection of a "meaningful" TF for a given dataset, as the features to be displayed depend highly on the user expectation. Early works attempted to find a good TF using clusters in histograms of statistical image properties [15] or fitting visibility histograms [8]. Others have focused on guiding an explorative user interaction [27, 59], also by using neural networks [3]. For optimizing a TF based on information measures, Ruiz *et al.* [43] proposed to bin voxels of similar density and match their visibility distribution from multiple viewpoints with a target distribution defined by local volume features. For optimization, the authors employ a gradient-based method where the visibility derivatives for each density bin are approximated via local linearization.

Concerning the performance of direct volume rendering, it is crucial to determine the minimum number of data samples that are required to accurately represent the volume. In prior works, strategies for optimal sampling in screen-space have been proposed, for instance, based on perceptual models [4], image saliency maps [39], entropy-based measures [56], temporal history [29], or using neural networks [53, 54]. Other approaches adaptively change the sampling stepsize along the view rays to reduce the number of samples in regions that do not contribute much to the image [6, 9, 23, 32]. DiffDVR's capability to compute gradients with respect to the stepsize gives rise to a gradient-based adaptation using image-based loss functions instead of gradient-free optimizations or heuristics.

Neural Rendering As an alternative to classical rendering techniques that are adapted to make them differentiable, several works have proposed to replace the whole rendering process with a neural network. For a general overview of neural rendering approaches let us refer to the recent summary article by Tewari *et al.* [48]. For example, RenderNet proposed by Nguyen-Phuoc *et al.* [34] replaces the mesh rasterizer with a combination of convolutional and fully connected networks. In visualization, the works by Berger *et al.* [3] and He *et al.* [16] fall into the same line of research. The former trained a network on rendered images and parameters of the rendering process, and use the network to predict new renderings by using only the camera and TF parameters. The latter let a network learn the relationships between the input parameters of a simulation and the rendered output, and then used this network to skip the rendering process and create images just from given input parameters.

3 Background

In the following, we review the fundamentals underlying DVR using an optical emission-absorption model [30]. Then we briefly summarise the foundation of Automatic Differentiation (AD), a method to systematically compute derivatives for arbitrary code [2].

[1] https://github.com/shamanDevel/DiffDVR

3.1 Direct Volume Rendering Integral

Let $V:\mathbb{R}^3\to[0,1]$ be the scalar volume of densities and let $r:\mathbb{R}^+\to\mathbb{R}^3$ be an arc-length parameterized ray through the volume. Let $\tau:[0,1]\to\mathbb{R}_0^+$ be the absorption and $C:[0,1]\to\mathbb{R}_0^+$ the self-emission due to a given density. Then, the light intensity reaching the eye is given by

$$L(a,b)=\int_a^b g(V(r(t)))e^{-\int_a^t \tau(V(r(u)))\mathrm{d}u}\mathrm{d}t, \tag{1}$$

were the exponential term is the transparency of the line segment from $t=a$, the eye, to b, the far plane, and $g(v)=\tau(v)C(v)$ is the emission. The transparency is one if the medium between a and b does not absorb any light and approaches zero for complete absorption.

We assume that the density volume is given at the vertices $\mathbf{v}_i$ of a rectangular grid, and the density values are obtained via *trilinear* interpolation. The functions τ and C define the mapping from density to absorption and emission. We assume that both functions are discretized into R regularly spaced control points with linear interpolation in between. This is realized on the GPU as a 1D texture map T with hardware-supported linear interpolation.

For arbitrary mappings of the density to absorption and emission, the volume rendering integral in Equation 1 cannot be solved analytically. Instead, it is approximated by discretizing the ray into N segments over which the absorption α_i and emission L_i are assumed constant. We make use of the Beer-Lambert model $\alpha_i=1-\exp(-\Delta t\,\tau(d_i))$, where d_i is the sampled volume density, to approximate a segment's transparency. This leads to a Riemann sum which can be computed in front-to-back order using iterative application of alpha-blending, i.e., $L=L+(1-\alpha)L_i$, and $\alpha=\alpha+(1-\alpha)\alpha_i$.

3.2 Automatic Differentiation

The evaluation of any program for a fixed input can be expressed as a computation graph, a directed acyclic graph where the nodes are the operations and the edges the intermediate values. Such a computation graph can be reformulated as a linear sequence of operations, also called a Wengert list [2, 55],

$$\begin{aligned}\boldsymbol{x}_0&=\text{const}\\ \boldsymbol{x}_1&=f_1(\boldsymbol{x}_0,\boldsymbol{w}_1)\\ \boldsymbol{x}_2&=f_2(\boldsymbol{x}_1,\boldsymbol{w}_2)\\ &\dots\\ \boldsymbol{x}_{\text{out}}&=f_k(\boldsymbol{x}_{k-1},\boldsymbol{w}_k)\end{aligned} \tag{2}$$

where the $\boldsymbol{w}_i$'s $\in\mathbb{R}^p$ are the external parameters of the operations of size p and the $\boldsymbol{x}_i$'s $\in\mathbb{R}^n$ refer to the state of intermediate results after the i-th operation of size n. The output $\boldsymbol{x}_{\text{out}}\in\mathbb{R}^m$ has size m. Note here that in DiffDVR, n and k are usually large, i.e., n is in the order of the number of pixels and k in the order of the number of sampling points along the view rays. The output $\boldsymbol{x}_{\text{out}}$ is a scalar ($m=1$), computed, for example, as the average per-pixel loss over the image. The goal is then to compute the derivatives $\frac{\mathrm{d}\boldsymbol{x}_{\text{out}}}{\mathrm{d}\boldsymbol{w}_i}$. The basic idea is to split these derivatives into simpler terms for each operation using the chain rule. For example, assuming univariate functions and w_1 the only parameter of interest, the chain rule yields

$$x_3=f_3(f_2(f_1(w_1)))\;\Rightarrow\;x_3'=f_3'(f_2(f_1(x)))\,f_2'(f_1(x))\,f_1'(x). \tag{3}$$

There are two fundamentally different approaches to automatically evaluate the chain rule, which depend on the order of evaluations. If the product in the above example is evaluated left-to-right, the derivatives are propagated from bottom to top in Equation 2. This gives rise to the adjoint- or backward-mode differentiation (see Sect. 4.3). If the product is evaluated right-to-left, the derivatives "ripple downward" from top to bottom in Equation 2. This corresponds to the so-called forward-mode differentiation (see Sect. 4.2).

4 AD for Direct Volume Rendering

Now we introduce the principal procedure when using AD for DiffDVR and hint at the task-dependent differences when applied for viewpoint optimization (Sect. 5.1), TF reconstruction (Sect. 5.2) and volume reconstruction (Sect. 5.3 and Sect. 5.4). We further discuss computational aspects and memory requirements of AD in volume rendering applications and introduce the specific modifications to make DiffDVR feasible.

4.1 The Direct Volume Rendering Algorithm

In direct volume rendering, the pixel color represents the accumulated attenuated emissions at the sampling points along the view rays. In the model of the Wengert list (see Equation 2), a function f_i is computed for each sample. Hence, the number of operations k is proportional to the overall number of samples along the rays. The intermediate results $\boldsymbol{x}_i$ are rgbα images of the rendered object up to the i-th sample, i.e., $\boldsymbol{x}_i$ is of size $n=\mathrm{W}*\mathrm{H}*4$, where W and H, respectively, are the width and height of the screen. The last operation f_k in the optimization process is the evaluation of a scalar-valued loss function. Thus, the size of the output variable is $m=1$. The parameters $\boldsymbol{w}_i$ depend on the use case. For instance, in viewpoint optimization, the optimization is for the longitude and latitude of the camera position, i.e., $p=2$. When reconstructing a TF, the optimization is for the R rgbα entries of the TF, i.e., $p=4R$.

The DVR algorithm with interpolation, TF mapping, and front-to-back blending is shown in Algorithm 1. For clarity, the variables in the algorithm are named by their function, instead of using $\boldsymbol{w}_i$ and $\boldsymbol{x}_i$ as in the Wengert list (Equation 2). In the Wengert list model, the step size Δt, the camera intrinsics *cam*, the TF T, and the volume density V are the parameters $\boldsymbol{w}_i$. The other intermediate variables are represented by the states $\boldsymbol{x}_i$. Each function operates on a single ray but is executed in parallel over all pixels.

Algorithm 1 Direct Volume Rendering Algorithm

Parameters: stepsize Δt, camera *cam*, TF T, volume V
Input: $\imath w$ the pixel positions where to shoot the rays

$color_i=0$ ▷ initial foreground color
$x_o,\boldsymbol{\omega}=f_{\text{camera}}(\imath w,cam)$ ▷ start x_o and direction $\boldsymbol{\omega}$ for all rays
for $i=0,\dots,N-1$ **do**
  $x_i=x_o+i\Delta t\,\boldsymbol{\omega}$ ▷ current position along the ray
  $d_i=f_{\text{interpolate}}(x_i,V)$ ▷ Trilinear interpolation
  $c_i=f_{\text{TF}}(d_i,T)$ ▷ TF evaluation
  $color_{i+1}=f_{\text{blend}}(color_i,c_i)$ ▷ blending of the sample
end for
$\boldsymbol{x}_{\text{out}}=f_{\text{loss}}(color_N)$ ▷ Loss function on the output rgbα image

When Algorithm 1 is executed, the operations form the computational graph. AD considers this graph to compute the derivatives of $\boldsymbol{x}_{\text{out}}$ with respect to the parameters $\Delta t, cam, T,$ and V, so that the changes that should be applied to the parameters to optimize the loss function can be computed automatically. Our implementation allows for computing derivatives with respect to all parameters, yet due to space limitations, we restrict the discussion to the computation of derivatives of $\boldsymbol{x}_{\text{out}}$ with respect to the camera *cam*, the TF T and the volume densities V. In the following, we discuss the concrete implementations of forward and adjoint differentiation to compute these derivatives.

4.2 Forward Differentiation

On the elementary level, the functions in Algorithm 1 can be expressed as a sequence of scalar arithmetic operations like $c=f(a,b)=a*b$. In forward-mode differentiation [2, 33], every variable is replaced by the associated *forward variable*

$$\tilde{a}=\left\langle a,\frac{\mathrm{d}a}{\mathrm{d}w}\right\rangle,\ \tilde{b}=\left\langle b,\frac{\mathrm{d}b}{\mathrm{d}w}\right\rangle, \tag{4}$$

i.e., tuples of the original value and the derivative with respect to the parameter w that is optimized. Each function $c=f(a,b)$ is replaced by the respective forward function

$$\tilde{c}=\tilde{f}(\tilde{a},\tilde{b})=\left\langle f(a,b),\frac{\partial f}{\partial a}\frac{\mathrm{d}a}{\mathrm{d}w}+\frac{\partial f}{\partial b}\frac{\mathrm{d}b}{\mathrm{d}w}\right\rangle. \tag{5}$$

Constant variables are initialized with zero, $\tilde{x}_{\text{const}} = \langle x_{\text{const}}, 0 \rangle$, and parameters for which to trace the derivatives are initialized with one, $\tilde{w} = \langle w, 1 \rangle$. If derivatives for multiple parameters should be computed, the tuple of forward variables is extended.

Forward differentiation uses a custom templated datatype for the forward variable and operator overloading. Each variable is wrapped in an instance of this datatype, called `fvar`, which stores the derivatives with respect to up to p parameters along with their current values.

```
template<typename T, int p>
struct fvar
{
    T value;
    T derivatives[p];
};
```

Next, operator overloads are provided for all common arithmetic operations and their gradients. For example, multiplication is implemented similar to:

```
template<typename T, int p>
fvar<T, p> operator*(fvar<T, p> a, fvar<T, p> b)
{
    fvar<T, P> c; //to store c = a*b and derivatives
    c.value = a.value * b.value;
    for (int i=0; i<p; ++i) { //partial derivatives
        c.derivative[i] = a.value*b.derivative[i]
            + b.value*a.derivative[i];
    }
    return c;
}
```

The user has to write the functions in such a way that arbitrary input types are possible, i.e., regular floats or instances of `fvar`, via C++ templates. All intermediate variables are declared with type `auto`. This allows the compiler to use normal arithmetic if no gradients are propagated, but when forward variables with gradients are passed as input, the corresponding operator overloads are chosen.

As an example (see Fig. 2 for a schematics), let us assume that derivatives should be computed with respect to a single entry in a 1D texture-based TF, e.g., the red channel of the first texel $T_{0,\text{red}}$. When loading the TF from memory, $T_{0,\text{red}}$ is replaced by $\tilde{T}_{0,\text{red}} = \langle T_{0,\text{red}}, 1 \rangle$, i.e., it is wrapped in an instance of `fvar` with the derivative for that parameter set to 1. Algorithm 1 executes in the normal way until $T_{0,\text{red}}$ is encountered in the code for the TF lookup. Now, the operator overloading mechanism selects the forward function instead of the normal non-differentiated function. The result is not a regular color c_i, but the forward variable of the color $\tilde{c}_i$. All following functions (i.e., the blend and loss function) continue to propagate the derivatives. In contrast, if derivatives should be computed with respect to the camera, already the first operation requires tracing the derivatives with `fvar`.

It is worth noting that in the above example only the derivative of one single texel in the TF is computed. This process needs to be repeated for each texel, respectively each color component of each texel, by extending the array `fvar::derivatives` to store the required number of p parameters. Notably, for input data that is high dimensional, like TFs or a 3D volumetric field, forward differentiation becomes unfeasible. For viewpoint selection, on the other hand, where only two parameters are optimized, forward differentiation can be performed efficiently.

The computational complexity of the forward method scales linearly with the number of parameters p, as they have to be propagated through every operation. However, as every forward variable directly stores the derivative of that variable w.r.t. the parameters, gradients for an arbitrary number of outputs m can be directly realized. Furthermore, the memory requirement is proportional to $O(np)$, as only the current state needs to be stored.

4.3 Adjoint Differentiation

Adjoint differentiation [31], also called the adjoint method, backward or reverse mode differentiation, or backpropagation, evaluates the chain

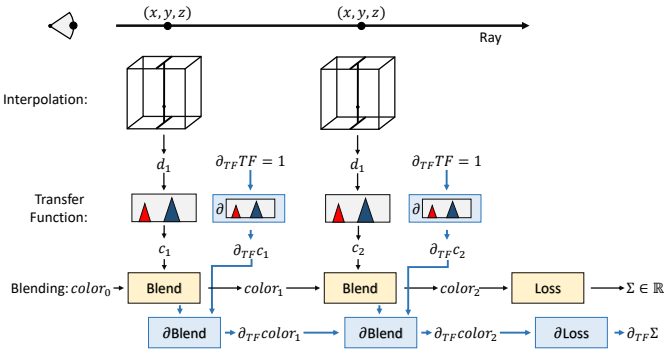

Fig. 2. Schematic representation of the forward method for TF reconstruction. Gradients are stored in the forward variables (blue), and parameter values are propagated simultaneously.

rule in the inverse order than forward differentiation. For each variable $\boldsymbol{x}_i$, the associated *adjoint variable*

$$\hat{\boldsymbol{x}}_i = \frac{\partial x_{\text{out}}}{\partial \boldsymbol{x}_i}\,,\quad \hat{\boldsymbol{w}}_i = \frac{\partial x_{\text{out}}}{\partial \boldsymbol{w}_i}, \tag{6}$$

stores the derivative of the final output with respect to the current variable. Tracing the derivatives starts by setting $\hat{x}_{\text{out}} = 1$. Then, the adjoint variables are tracked backward through the algorithm, called the *backward pass*. This is equivalent to evaluating the chain rule Equation 3 from left to right, instead of right to left as in the forward method. Let $c = f(a,b)$ be again our model function, then the adjoint variables $\hat{a}, \hat{b}$ are computed from $\hat{c}$ as

$$\hat{a} = \left(\frac{\partial f}{\partial a}\right)^T \hat{c},\ \hat{b} = \left(\frac{\partial f}{\partial b}\right)^T \hat{c}. \tag{7}$$

This process is repeated from the last operation to the first operation, giving rise to the *adjoint code*. At the end, one arrives again at the derivatives with respect to the parameters $\hat{\boldsymbol{w}} = \frac{\partial x_{\text{out}}}{\partial \boldsymbol{w}}$. If a parameter is used multiple times, either along the ray or over multiple rays, the adjoint variables are summed up. The reverted evaluation of the DVR algorithm with the gradient propagation from Equation 7 is sketched in Algorithm 2. A schematic visualization is shown in Fig. 3.

Because the adjoint method requires reversing the order of operation, simple operator overloading as in the forward method is no longer applicable. Common implementations of the adjoint method like TensorFlow [1] or PyTorch [40] record the operations in a computation graph, which is then traversed backward in the backward pass. As it is too costly to record every single arithmetic operation, high-level

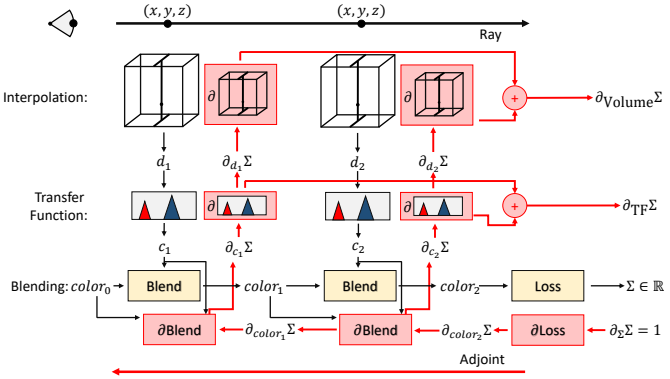

Fig. 3. Schematic representation of the adjoint method for density and TF reconstruction. Gradients in the adjoint variables (red) are propagated backward through the algorithm. A circled + indicates the summation of the gradients over all steps and rays.

Algorithm 2 Adjoint Code of the DVR Algorithm. Each line corresponds to a line in Algorithm 1 in reverse order.

Parameters: stepsize Δt, camera cam, TF T, volume V
Input: the adjoint of the output $\hat{\boldsymbol{x}}_{\text{out}}$
all intermediate adjoint variables are initialized with 0

1: $\hat{color}_N \mathrel{+}= \partial f_{\text{loss}}(color_N)^T \hat{\boldsymbol{x}}_{\text{out}}$
2: **for** $i = N-1, \ldots, 0$ **do**
3: $\quad \hat{color}_i, \hat{c}_i \mathrel{+}= \partial f_{\text{blend}}(color_i, c_i)^T \hat{color}_N$
4: $\quad \hat{d}_i, \hat{T} \mathrel{+}= \partial f_{\text{TF}}(d_i, T)^T \hat{c}_i$
5: $\quad \hat{x}_i, \hat{V} \mathrel{+}= \partial f_{\text{interpolate}}(x_i, V)^T \hat{d}_i$
6: $\quad \hat{x}_o \mathrel{+}= \hat{x}_i\ ,\ \hat{\Delta t} \mathrel{+}= i\omega^T \hat{x}_i\ ,\ \hat{\omega} \mathrel{+}= i\Delta t \hat{x}_i$
7: **end for**
8: $\hat{cam} \mathrel{+}= \partial f_{\text{camera}}(\imath w, cam)^T [x_o; \omega]$
9: $\hat{color}_0$ is ignored

Output: $\hat{\Delta t}, \hat{cam}, \hat{T}, \hat{V}$

functions like the evaluation of a single layer in neural networks are treated as atomic, and only these are recorded. Within such a high-level function, the order of operations is known and the adjoint code using Equation 7 is manually derived and implemented. We follow the same idea and treat the rendering algorithm as one unit and manually derive the adjoint code.

4.4 The Inversion Trick

One of the major limitations of the adjoint method is its memory consumption because the input values for the gradient computations need to be stored. For example, the blending operation (line 7 in Algorithm 1) is defined as follows: Let α, C be the opacity and rgb-emission at the current sample, i.e., the components of c_i, and let $\alpha^{(i)}, C^{(i)}$ be the accumulated opacity and emission up to the current sample, i.e., the components of $color_i$ in Algorithm 1. Then, the next opacity and emission is given by front-to-back blending

$$\begin{aligned} C^{(i+1)} &= C^{(i)} + (1-\alpha^{(i)})C \\ \alpha^{(i+1)} &= \alpha^{(i)} + (1-\alpha^{(i)})\alpha. \end{aligned} \tag{8}$$

In the following adjoint code with $\hat{\alpha}^{(i+1)}, \hat{C}^{(i+1)}$ as input it can be seen that the derivatives again require the input values.

$$\begin{aligned} \hat{\alpha} &= (1-\alpha^{(i)})\hat{\alpha}^{(i+1)},\ \hat{C} = (C-\alpha^{(i)})\hat{C}^{(i+1)}, \\ \hat{\alpha}^{(i)} &= (1-\alpha)\hat{\alpha}^{(i+1)} - C \cdot \hat{C}^{(i+1)}, \\ \hat{C}^{(i)} &= \hat{C}^{(i+1)}. \end{aligned} \tag{9}$$

Therefore, the algorithm is first executed in its non-adjoint form, and the intermediate colors are stored with the computation graph. This is called the *forward pass*. During the backward pass, when the order of operations is reversed and the derivatives are propagated (the adjoint code), the intermediate values are reused. In DVR, intermediate values need to be stored at every step through the volume. Thus, the memory requirement scales linearly with the number of steps and quickly exceeds the available memory. To overcome this limitation, we propose a method that avoids storing the intermediate colors after each step and, thus, has a constant memory requirement.

We exploit that the blending step is invertible (see Fig. 4): If $\alpha^{(i+1)}, C^{(i+1)}$ are given and the current sample is recomputed to obtain α and C, $\alpha^{(i)}, C^{(i)}$ can be reconstructed as

$$\begin{aligned} \alpha^{(i)} &= \frac{\alpha - \alpha^{(i+1)}}{\alpha - 1} \\ C^{(i)} &= C^{(i+1)} - (1-\alpha^{(i)})C. \end{aligned} \tag{10}$$

With Equation 10 and $\alpha < 1$, the adjoint pass can be computed with constant memory by re-evaluating the current sample c_i and reconstructing $color_i$ instead of storing the intermediate results. Thus, only the output color used in the loss function needs to be stored, while all intermediate values are recomputed on-the-fly. Note that $\alpha = 1$ is not possible in practice, since it requires the absorption stored in the TF to be at infinity.

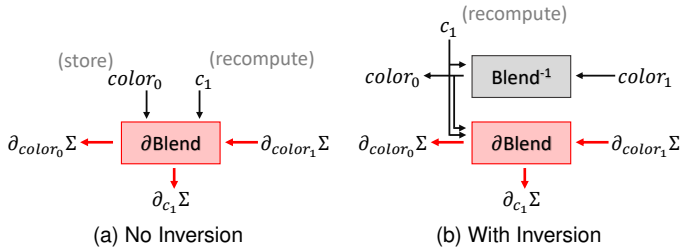

Fig. 4. (a) To compute the current contribution c_i, intermediate accumulated colors $color_i$ need to be stored for every step along the ray. (b) The inversion trick enables to reconstruct $color_i$ from $color_{i+1}$. Thus, only the final color used in the loss function needs to be stored.

In the implementation, and indicated by the circled + in Fig. 3, the adjoint variables for the parameters are first accumulated per ray into local registers (camera, stepsize, volume densities) or shared memory (TF). Then, the variables are accumulated over all rays using global atomic functions. This happens once all rays have been traversed (camera, stepsize, transfer function) or on exit of the current cell (volume densities).

Because the adjoint variables carry only the derivatives of the output, but not of the parameters, the computational complexity is largely constant in the number of parameters. For example, in TF optimization (Sect. 5.2) only the derivative of the currently accessed texel is computed when accessed in the adjoint code of TF sampling. This is significantly different from the forward method, where the derivatives of all TF entries need to be propagated in every step. On the other hand, the adjoint method considers only a single scalar output in each backward pass, requiring multiple passes to support multi-component outputs. This analysis and the following example applications show that the forward method is preferable when optimizing for a low number of parameters like the camera position, while for applications such as TF optimization, which require the optimization of many parameters, the adjoint method has clear performance advantages.

DiffDVR is implemented as a custom CUDA operation in PyTorch [40]. The various components of the DVR algorithm, like the parameter to differentiate or the type of TF, are selected via C++ template parameters. This eliminates runtime conditionals in the computation kernel. To avoid pre-compiling all possible combinations, the requested configuration is compiled on demand via CUDA's JIT-compiler NVRTC [38] and cached between runs. This differs from, e.g., the Enoki library [18] used by the Mitsuba renderer [37], which directly generates Parallel Thread Code (PTX) for translation into GPU binary code.

5 Applications

In the following, we apply both AD modes for best viewpoint selection, TF reconstruction, and volume reconstruction. The results are analyzed both qualitatively and quantitatively. Timings are performed on a system running Windows 10 and CUDA 11.1 with an Intel Xeon 8x@3.60Ghz CPU, 64GB RAM, and an NVIDIA RTX 2070.

5.1 Best Viewpoint Selection

We assume that the camera is placed on a sphere enclosing the volume and faces toward the object center. The camera is parameterized by longitude and latitude. AD is used to optimize the camera parameters to determine the viewpoint that maximized the selected cost function. As cost function, we adopt the differentiable opacity entropy proposed by Ji *et al.* [19].

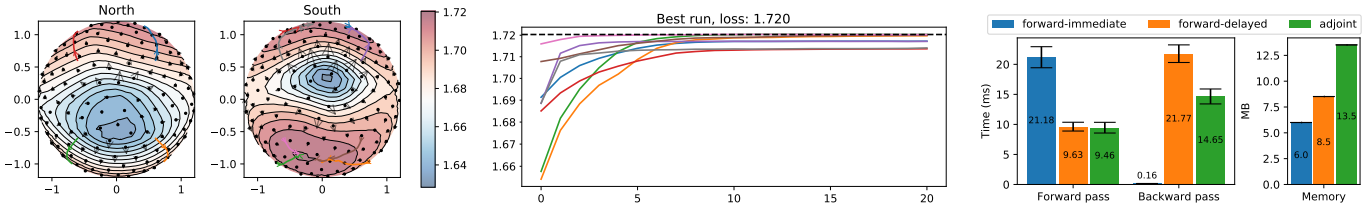

Fig. 5. Best viewpoint selection using maximization of visual entropy. The tooth dataset (Fig. 1a) is rendered from different viewpoints on a surrounding sphere. (a) Color coding of loss values for viewpoints on the northern and southern hemispheres, with isocontours (black lines) of the loss and local gradients with respect to the longitude and latitude of the camera position at uniformly sampled positions (black dots with arrows). Eight optimization runs (colored paths on the surface) are started at uniformly seeded positions and optimized in parallel. (b) The runs converge to three clusters of local minima. The cluster with the highest entropy (1.72) coincides with the best value from 256 sampled entropies. For the best run, the start view, as well as some intermediate views and the final result, are shown in Fig. 1a. (c) Timings and memory consumption show that forward differences approximately double the runtime, but are faster and require less memory than the adjoint method.

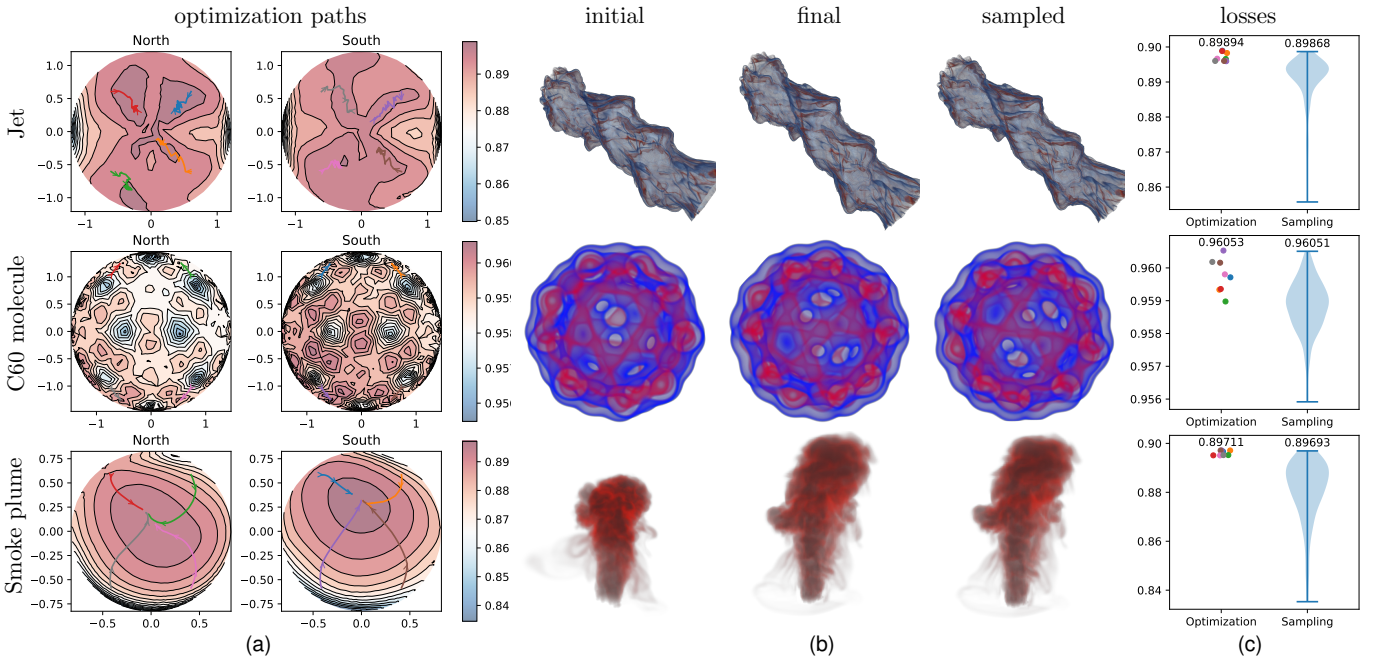

Fig. 6. Best viewpoint selection using maximization of visual entropy for datasets jetstream (256^3), potential field of a C60 molecule (128^3), and smoke plume (178^3). Comparison of DiffDVR with eight initializations against random uniform sampling over the sphere of 256 views. (a) Optimization paths over the sphere. (b) Initial view, selected view of DiffDVR, best sampled view. (c) Visual entropy of optimization results (colored points corresponding to (a)) vs. sampled images. Violin plot shows the distribution of loss values when sampling the sphere uniformly. Visual entropy of the best viewport is shown above each plot.

Let $C \in \mathbb{R}^{H\times W\times 4}$ be the output image. We employ array notation, i.e., $C[x,y,c]$ indicates color channel c (red, green, blue, alpha) at pixel x,y. The entropy of a vector $\boldsymbol{x} \in \mathbb{R}^N$ is defined as

$$H(\boldsymbol{x}) = \frac{1}{\log_2 N} \sum_{i=1}^{N} p_i \log_2 p_i \,, \quad p_i = \frac{\boldsymbol{x}_i}{\sum_{j=1}^{N} \boldsymbol{x}_j}. \tag{11}$$

Then the opacity entropy is defined as $\mathrm{OE}(C) = H(C[:,:,3])$, where $C[:,:,3]$ indicates the linearization of the alpha channel, and the color information is unused.

In a first experiment, the best viewpoint is computed for a CT scan of a human tooth of resolution $256 \times 256 \times 161$. Eight optimizations are started in parallel with initial views from viewpoints at a longitude of $\{45°, 135°, 225°, 315°\}$ and a latitude of $\pm 45°$. In all cases, 20 iterations using gradient descent are performed. The viewpoints selected by the optimizer are shown as paths over the sphere in Fig. 5a. The values of the cost function over the course of optimization are given in Fig. 5b. It can be seen that the eight optimization runs converge to three distinct local minima. The best run converges to approximately the same entropy as obtained when the best view from 256 uniformly sampled views over the enclosing sphere is taken. Fig. 1a shows intermediate views and the view from the optimized viewpoint. Further results on other datasets, i.e, a jetstream simulation (256^3), the potential field of a C60 molecule (128^3), and a smoke plume (178^3), confirm the dependency of the optimization process on the initial view (see Fig. 6).

Both the adjoint and the forward method compute exactly the same gradients, except for rounding errors. As seen in Fig. 5c, a single forward/backward pass in the adjoint method requires about 9.5ms/14.6ms, respectively, giving a total of 24.1ms. For the forward method, we compare two alternatives. First, *forward-immediate* directly evaluates the forward variables during the forward pass in PyTorch and stores these variables for reuse in the backward pass. In *forward-delayed*, the evaluation of gradients is delayed until the backward pass, requiring to re-trace the volume. With 21.3ms, *forward-immediate* is slightly faster than the adjoint method, while *forward-delayed* is around 30% slower due to the re-trace operation.

5.2 Transfer Function Reconstruction

Our second use case is TF reconstruction. Reference images of a volume are first rendered from multiple views using a target TF. Given

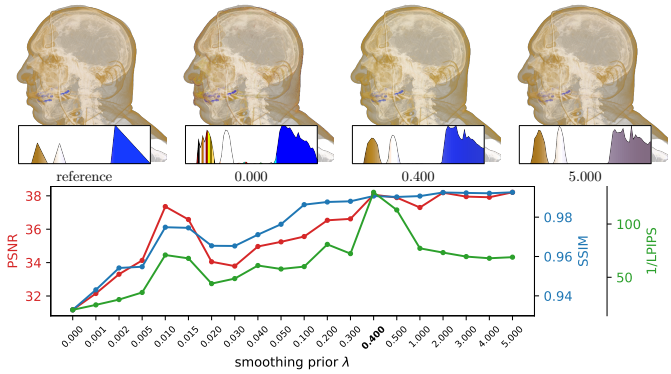

Fig. 7. Effect of the smoothing prior (Equation 12). A small value of λ leads to "jagged" TFs which can accurately predict small details like the teeth in blue but introduce low frequency color shifts resulting in low PSNR and SSIM [52]. A large smoothing prior smooths out small details.

the same volume and an arbitrary initial TF, AD is then used to optimize the TF so that the rendered images match the references. The target TF comprises of 256 rgbα entries, the target TF with R entries is initialized with random Gaussian noise. The density volume is rendered to a 512^2 viewport from eight camera positions that are uniformly distributed around the volume (the view direction always pointing toward the volume's center).

Let $T \in \mathbb{R}^{R,4}$ be the TF with R entries containing the rgb color and absorption, and let $\boldsymbol{x}_i$ be the N rendered image of resolution $W \times H$, $\boldsymbol{y}_i$ are the reference images. In our case, $N = 8, W = H = 512$ and R varies. We employ an L_1 loss on the images and a smoothing prior L_{prior} on the TF, i.e.,

$$\begin{aligned} L_{\text{total}} &= L_1(\boldsymbol{x}) + \lambda L_{\text{prior}}(T), \\ L_1(\boldsymbol{x}) &= \frac{1}{NWH} \sum_{i,x,y} |\boldsymbol{x}_{ixy} - \boldsymbol{y}_{ixy}| \\ L_{\text{prior}}(T) &= \frac{1}{4(R-1)} \sum_{c=1}^{4} \sum_{r=1}^{R-1} (T_{c,r+1} - T_{c,r})^2. \end{aligned} \tag{12}$$

The Adam optimizer [22] is used with a learning rate of 0.8 for 200 epochs. The use of λ to control the strength of the smoothing prior is demonstrated in Fig. 7 for a human head CT scan as test dataset using $R = 64$. If λ is too small, the reconstructed TF contains high frequencies and introduces subtle color shifts over the whole image. If the smoothing prior is too large, small details are lost. We found that a value of λ around 0.4 leads to the best results, visually and using the Learned Perceptual Image Patch Similarity metric (LPIPS) [58], and is thus used in our experiments. We chose the LPIPS metric as we found that it can accurately distinguish the perceptually best results when the peak-signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) [52] result in similar scores. The initialization of the reconstruction and the final result for a human head dataset are shown in Fig. 1b.

Next, we analyze the impact of the TF resolution R on reconstruction quality and performance (see Fig. 8). For TF reconstruction, the backward AD mode significantly outperforms the forward mode. Because of the large number of parameters, especially when increasing the resolution of the TF, the derivatives of many parameters have to be traced in every operation when using the forward AD mode. Furthermore, the forward variables may no longer fit into registers and overflow into global memory. This introduces a large memory overhead that leads to a performance decrease that is even worse than the expected linear decrease. A naïve implementation of the adjoint method that directly accumulates the gradients for the TF in global memory using atomics is over 100× slower than the non-adjoint forward pass (*adjoint-immediate*). This is because of the large number of memory accesses and write conflicts. Therefore, we employ delayed accumulation (*adjoint-delayed*). The gradients for the TF are first accumulated

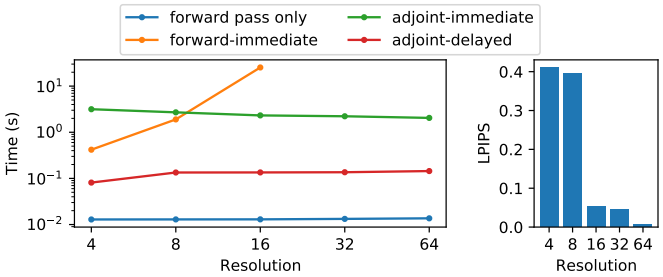

Fig. 8. Timings and loss function values for different AD modes and resolutions of the reconstructed TF. Timings are with respect to a single epoch.

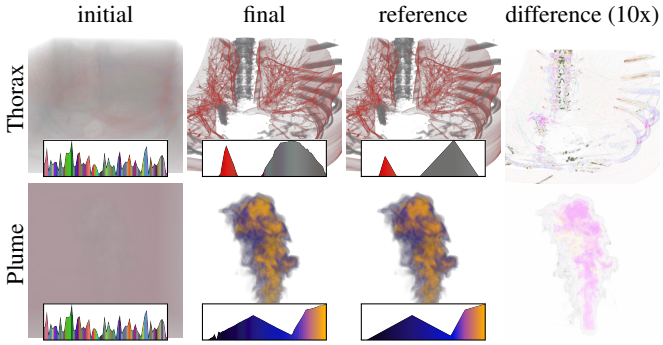

Fig. 9. TF reconstruction using a CT scan of a human thorax (PSNR=42.6dB) and a smoke plume (PSNR=47.8dB, SSIM=0.999). The used hyperparameters are the same as for the skull dataset. From left to right: Start configurations for the optimizer, optimized results, ground truths, pixel differences (scaled by a factor of 10 for better perception) between ground truths and optimized results.

in shared memory. Then, after all threads have finished their assigned rays, the gradients are reduced in parallel and then accumulated into global memory using atomics. As seen in Fig. 8, this is the fastest of the presented methods. The whole optimization for 200 epochs requires around 5 minutes including I/O. However, as only 48kB of shared memory are freely available per multiprocessor, the maximal resolution of the TF is 96 texels. If a higher resolution is required, *adjoint-immediate* must be employed. At smaller values of R the reconstruction quality is decreased (see Fig. 8). We found that a resolution of $R = 64$ leads to the best compromise between reconstruction performance and computation time.

To evaluate the capabilities of TF reconstruction to generalize to new datasets with the same hyperparameters as described above, we run the optimization on two new datasets, a CT scan of a human thorax and a smoke plume, both of resolution 256^3. As one can see in Fig. 9, the renderings with the reconstructed TF closely match the reference, demonstrating stability of the optimization for other datasets.

We envision that TF optimization with respect to losses in screen space can be used to generate "good" TFs for a dataset for which no TF is available. While a lot of research has been conducted on measuring the image quality for viewpoint selection, quality metrics specialized for TFs are still an open question to the best of our knowledge. In future work, a first approach would be to take renderings of other datasets with a TF designed by experts and transform the "style" of that rendering to a new dataset via the style loss by Gatys *et al.* [11].

5.3 Density Reconstruction

In the following, we shed light on the use of DiffDVR for reconstructing a 3D density field from images of this field. For pure absorption models, the problem reduces to a linear optimization problem. This allows for comparisons with specialized methods, such as filtered backpropagation or algebraic reconstruction [10, 13, 17]. We compare DiffDVR to the CUDA implementation of the SIRT algebraic reconstruction algo-

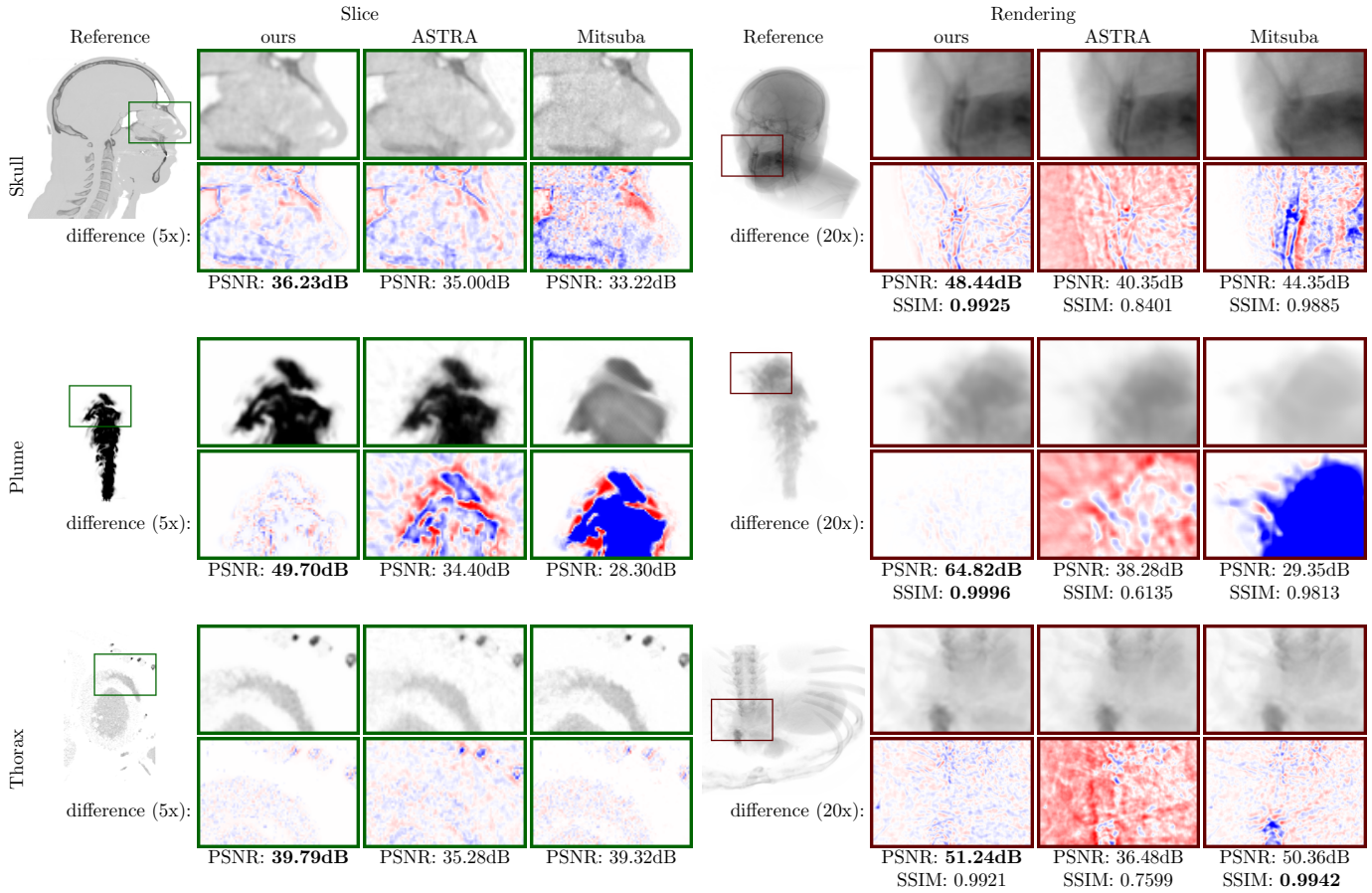

Fig. 10. Density reconstruction using an optical absorption-only model. Comparison between DiffDVR, algebraic reconstruction provided by the ASTRA-toolbox [49, 50] and Mitsuba's differentiable path tracer [36]. For each algorithm, a single slice through the center of the reconstructed volume and a volume rendering of this volume are shown, including per-pixel differences to the reference images. PSNR values in column "slice" are computed over the whole volume, in column "rendering" they are with respect to the rendered images. Timings are given in Sect. 5.3. In the difference images, blue and red indicate under- and over-estimation, respectively.

rithm [14] provided by the ASTRA-toolbox [49, 50]. Furthermore, we compare the results to those computed by Mitsuba 2 [36, 37], a general differentiable path tracer. Density reconstruction uses 64 uniformly sampled views on a surrounding sphere. Each image is rendered at a resolution of 512^2. The reconstructed volume has a resolution of 256^3. ASTRA and Mitsuba are used with their default optimization settings. DiffDVR performs a stepsize of 0.2 voxels during reconstruction. The Adam optimizer with a batch size of 8 images and a learning rate of 0.3 is used. To speed up convergence, we start with a volume of resolution 32^3 and double the resolution in each dimension after 10 iterations. At the highest resolution, the optimization is performed for 50 iterations. The same L_{total} loss function as for TF reconstruction (see Equation 12) is used, except that the smoothing prior is computed on the reconstructed volume densities in 3D, with $\lambda = 0.5$.

Three experiments with datasets exhibiting different characteristics are carried out. The results are shown in Fig. 10. As one can see, DiffDVR consistently outperforms algebraic reconstruction via ASTRA and density reconstruction via the Mitsuba framework. In particular, Mitsuba suffers from noise in the volume due to the use of stochastic path tracing. Only for the rendering of the thorax dataset, Mitsuba shows a slightly better SSIM score than DiffDVR. For the plume dataset, intermediate results of the optimization process until convergence are shown in Fig. 1c.

Note that all compared algorithms serve different purposes. Algebraic reconstruction methods (ASTRA) are specialized for absorption-only optical models and support only such models. Mitsuba is tailored to Monte Carlo path tracing with volumetric scattering, an inherently computational expensive task. DifffDVR is specialized for direct volume rendering with an emission-absorption model and a TF, yet emissions and a TF were disabled in the current experiments. These differences clearly reflect in the reconstruction times. For instance, for reconstructing the human skull dataset, ASTRA requires only 53 seconds, DiffDVR requires around 12 minutes, and Mitsuba runs for multiple hours.

5.4 Color Reconstruction

Next, we consider an optical emission-absorption model with a TF that maps densities to colors and opacities, as it is commonly used in DVR. To the best of our knowledge, we are the first to support such a model in tomographic reconstruction.

For TFs that are not a monotonic ramp, as in the absorption-only case, density optimization becomes a non-convex problem. Therefore, the optimization can be guided into local minima by a poor initialization. We illustrate this problem in a simple 1D example. A single unknown density value d_1 of a 1D "voxel" – a line segment with two values $d_0 = -1$ and d_1 at the end points and linear interpolation in between – should be optimized. A single Gaussian function with zero mean and variance 0.5 is used as TF, and the ground truth value for d_1 is -1. For varying d_1, Fig. 12 shows the L_2-loss between the color obtained from d_1 and the ground truth, and the corresponding gradients. As can be seen, for initial values of $d_1 > 0.4$ the gradient points away from the true solution. Thus, the optimization "gets stuck" at the other side of the Gaussian, never reaching the target density of -1. This issue worsens in 2D and 3D, as the optimizer needs to reconstruct a globally consistent density field considering many local constraints.

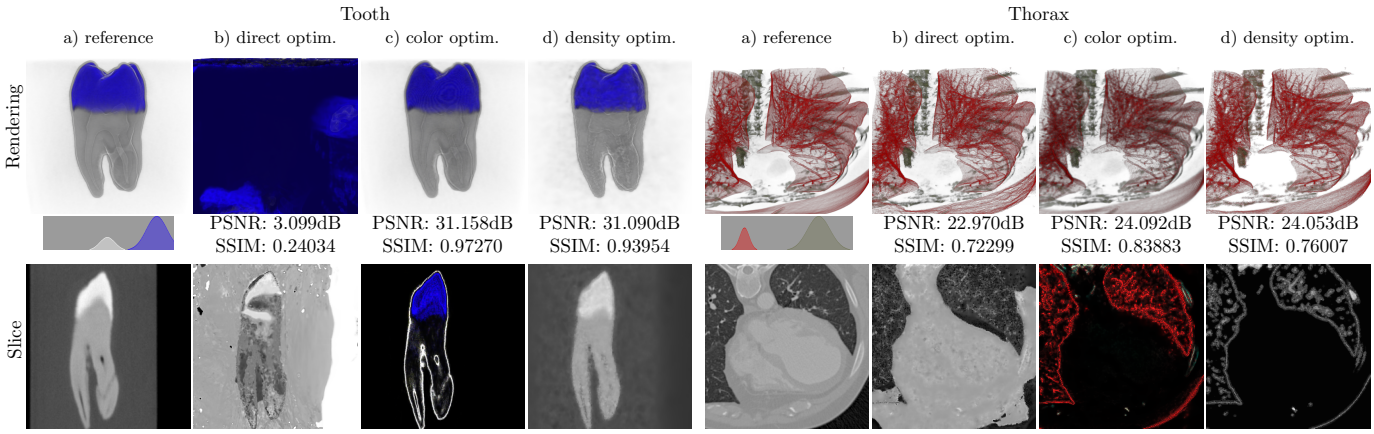

Fig. 11. Density optimization for a volume colored via a non-monotonic rgbα-TF using an emission-absorption model. (a) Rendering of the reference volume of a human tooth and a human thorax. (b) Local minimum of the loss function. (c) Pre-shaded color volume as initialization. (d) Final result of the density volume optimization with TF mapping. The second row shows slices through the volumes. Note the colored slice through the pre-shaded color volume in (c).

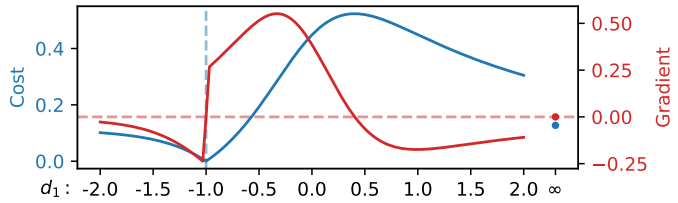

Fig. 12. 1D example for a density optimization with a Gaussian TF with the optimum at a density of -1.0. For a value > 0.4, the gradient faces away from the optimum.

This failure case is also shown in Fig. 11b, where the tooth dataset cannot be reconstructed faithfully due to the initialization with a poorly matching initial field.

To overcome this shortcoming, it is crucial to start the optimization with an initial guess that is not "too far" from the ground truth in the high-dimensional parameter space. We account for this by proposing the following optimization pipeline: First, a pre-shaded color volume of resolution 256^3 (Fig. 11c) is reconstructed from images using the same multi-resolution optimization as in the case of an absorption-only model. The color volume stores the rgb-emission and scalar absorption per voxel, instead of a scalar density value that is mapped to color via a TF. By using this color volume, trapping into local minima with non-monotonic TFs can be avoid. Intermediate results of the optimization process until convergence are shown in Fig. 1d for the tooth dataset. Then, density values that match the reconstructed colors after applying the TF are estimated. For each voxel, 256 random values are sampled, converted to color via the TF, and the best match is chosen. To avoid inconsistencies between neighboring voxels, an additional loss term penalises differences to neighbors. Let (τ_T, C_T) be the target color from the color volume and d the sampled density with mapped color $\tau(d), C(d)$, then the cost function is

$$\mathscr{C}(d) = ||C_T - C(d)||_2^2 + \alpha \log(1 + |\tau_T - \tau(d)|) + \beta \sum_{i \in \mathscr{N}} (d - d_i)^2. \tag{13}$$

Here, α and β are weights, and $\mathscr{N}$ loops over the 6-neighborhood of the current voxel. The logarithm accounts for the vastly different scales of the absorption, similar to an inverse of the transparency integral Equation 1. In the example, we set $\alpha = 1/\max(\tau_T)$ to normalize for the maximal absorption in the color volume, and $\beta = 1$. This process is repeated until the changes between subsequent iterations fall below a certain threshold, or a prescribed number of iterations have been performed.

Finally, the estimated density volume is used as initialization for the optimization of the density volume from the rendered images (Fig. 11d). We employ the same loss L_{total} as before with a smoothing prior of $\lambda = 20$. The total runtime for a 256^3 volume is roughly 50 minutes. Even though the proposed initialization overcomes to a certain extent the problem of non-convexity and yields reasonable results, Fig. 11 indicates that some fine details are lost and spurious noise remains. We attribute this to remaining ambiguities in the sampling of densities from colors that still lead to suboptimal minima in the reconstruction. This also shows in the slice view of Fig. 11d, especially for the thorax dataset. Here, some areas that are fully transparent due to the TF are arbitrarily mapped to a density value of zero, while the reference has a density around 0.5 – between the peaks of the TF – in these areas.

6 Conclusion

In this work, we have introduced a framework for differentiable direct volume rendering (DiffDVR), and we have demonstrated its use in a number of different tasks related to data visualization. We have shown that differentiability of the direct volume rendering process with respect to the viewpoint position, the TF, and the volume densities is feasible, and can be performed at reasonable memory requirements and surprisingly good performance.

Our results indicate the potential of the proposed framework to automatically determine optimal parameter combinations regarding different loss functions. This makes DiffDVR in particular interesting in combination with neural networks. Such networks might be used as loss functions – providing blackboxes, which steer DiffDVR to an optimal output for training purposes, e.g., to synthesize volume-rendered imagery for transfer learning tasks. Furthermore, derivatives with respect to the volume from rendered images promise the application to scene representation networks trained in screen space instead of from points in object space. We see this as one of the most interesting future works, spawning future research towards the development of techniques that can convert large data to a compact representation -- a code -- that can be permanently stored and accessed by a network-based visualization. Besides neural networks, we imagine possible applications in the development of lossy compression algorithms, e.g. via wavelets, where the compression rate is not determined by losses in world space, but by the quality of rendered images. The question we will address in the future is how to generate such (visualization-)task-dependent codes that can be intertwined with differentiable renderers.

Acknowledgments

The authors wish to thank Jakob Wenzel and Merlin Nimier-David for their help and valuable suggestions on the Mitsuba 2 framework.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[2] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124(1):171–190, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations. doi: 10.1016/S0377-0427(00)00422-2

[3] 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, 2018.

[4] M. R. Bolin and G. W. Meyer. A perceptually based adaptive sampling algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, p. 299–309. Association for Computing Machinery, New York, NY, USA, 1998. doi: 10.1145/280814.280924

[5] U. D. Bordoloi and H.-W. Shen. View selection for volume rendering. In *VIS 05. IEEE Visualization, 2005.*, pp. 487–494. IEEE, 2005.

[6] L. Q. Campagnolo, W. Celes, and L. H. de Figueiredo. Accurate volume rendering based on adaptive numerical integration. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pp. 17–24. IEEE, 2015.

[7] M. Chen and H. Jäenicke. An information-theoretic framework for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1206–1215, 2010.

[8] C. D. Correa and K.-L. Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):192–204, 2010.

[9] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS '92, p. 91–98. Association for Computing Machinery, New York, NY, USA, 1992. doi: 10.1145/147130.147155

[10] D. Dudgeon, R. Mersereau, and R. Merser. Multidimensional digital signal processing. prentice hall. *Englewood Cliffs, NJ*, 19842, 1984.

[11] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423, 2016.

[12] I. Gkioulekas, S. Zhao, K. Bala, T. Zickler, and A. Levin. Inverse volume rendering with material dictionaries. *ACM Transactions on Graphics (TOG)*, 32(6):1–13, 2013.

[13] R. Gordon, R. Bender, and G. T. Herman. Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *Journal of Theoretical Biology*, 29(3):471–481, 1970. doi: 10.1016/0022-5193(70)90109-8

[14] J. Gregor and T. Benson. Computational analysis and improvement of sirt. *IEEE Transactions on Medical Imaging*, 27(7):918–924, 2008. doi: 10.1109/TMI.2008.923696

[15] M. Haidacher, D. Patel, S. Bruckner, A. Kanitsar, and M. E. Gröller. Volume visualization based on statistical transfer-function spaces. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 17–24. IEEE, 2010.

[16] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. 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, 2019.

[17] G. T. Herman. *Fundamentals of computerized tomography: image reconstruction from projections*. Springer Science & Business Media, 2009.

[18] W. Jakob. Enoki: structured vectorization and differentiation on modern processor architectures, 2019. `https://github.com/mitsuba-renderer/enoki`.

[19] G. Ji and H.-W. Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[20] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

[21] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3907–3916, 2018.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] A. Kratz, J. Reininghaus, M. Hadwiger, and I. Hotz. Adaptive screen-space sampling for volume ray-casting. *ZIB-Report*, 2011.

[24] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[25] S. Liu, S. Saito, W. Chen, and H. Li. Learning to infer implicit surfaces without 3d supervision. *NeurIPS*, 2019.

[26] M. M. Loper and M. J. Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pp. 154–169. Springer, 2014.

[27] R. Maciejewski, Y. Jang, I. Woo, H. Jänicke, K. P. Gaither, and D. S. Ebert. Abstracting attribute space for transfer function exploration and design. *IEEE Transactions on Visualization and Computer Graphics*, 19(1):94–107, 2012.

[28] R. Marques, C. Bouville, M. Ribardière, L. P. Santos, and K. Bouatouch. Spherical fibonacci point sets for illumination integrals. *Computer Graphics Forum*, 32(8):134–143, 2013. doi: 10.1111/cgf.12190

[29] J. Martschinke, S. Hartnagel, B. Keinert, K. Engel, and M. Stamminger. Adaptive temporal sampling for volumetric path tracing of medical data. *Computer Graphics Forum*, 38(4):67–76, 2019. doi: 10.1111/cgf.13771

[30] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[31] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, Aug. 2004. doi: 10.1145/1015706.1015744

[32] N. Morrical, W. Usher, I. Wald, and V. Pascucci. Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing. In *2019 IEEE Visualization Conference (VIS)*, pp. 256–260, 2019. doi: 10.1109/VISUAL.2019.8933539

[33] R. D. Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[34] T. Nguyen-Phuoc, C. Li, S. Balaban, and Y.-L. Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. *arXiv preprint arXiv:1806.06575*, 2018.

[35] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[36] M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4), July 2020. doi: 10.1145/3386569.3392406

[37] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), Nov. 2019. doi: 10.1145/3355089.3356498

[38] NVidia. Cuda nvrtc, 2021. `https://docs.nvidia.com/cuda/nvrtc/index.html`.

[39] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pp. 281–288, 1989.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[41] F. Petersen, A. H. Bermano, O. Deussen, and D. Cohen-Or. Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer. *arXiv preprint arXiv:1903.11149*, 2019.

[42] H. Rhodin, N. Robertini, C. Richardt, H.-P. Seidel, and C. Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 765–773, 2015.

[43] M. Ruiz, A. Bardera, I. Boada, and I. Viola. Automatic transfer functions based on informational divergence. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1932–1941, 2011.

[44] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *NeurIPS*, 2019.

[45] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *VIS*

05. IEEE Visualization, 2005., pp. 495–502. IEEE, 2005.

[46] Y. Tao, H. Lin, H. Bao, F. Dong, and G. Clapworthy. Structure-aware viewpoint selection for volume visualization. In *2009 IEEE Pacific Visualization Symposium*, pp. 193–200. IEEE, 2009.

[47] Y. Tao, Q. Wang, W. Chen, Y. Wu, and H. Lin. Similarity voting based viewpoint selection for volumes. In *Computer graphics forum*, vol. 35, pp. 391–400. Wiley Online Library, 2016.

[48] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, vol. 39, pp. 701–727. Wiley Online Library, 2020.

[49] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. D. Beenhouwer, K. J. Batenburg, and J. Sijbers. Fast and flexible x-ray tomography using the astra toolbox. *Opt. Express*, 24(22):25129–25147, Oct 2016. doi: 10.1364/OE.24.025129

[50] W. van Aarle, W. J. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. J. Batenburg, and J. Sijbers. The astra toolbox: A platform for advanced algorithm development in electron tomography. *Ultramicroscopy*, 157:35–47, 2015. doi: 10.1016/j.ultramic.2015.05.002

[51] P.-P. Vázquez, E. Monclús, and I. Navazo. Representative views and paths for volume models. In *International Symposium on Smart Graphics*, pp. 106–117. Springer, 2008.

[52] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[53] 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*, pp. 1–1, 2019. doi: 10.1109/TVCG.2019.2956697

[54] S. Weiss, M. Işık, J. Thies, and R. Westermann. Learning adaptive sampling and reconstruction for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020. doi: 10.1109/TVCG.2020.3039340

[55] R. E. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.

[56] Q. Xu, S. Bao, R. Zhang, R. Hu, and M. Sbert. Adaptive sampling for monte carlo global illumination using tsallis entropy. In *International Conference on Computational and Information Science*, pp. 989–994. Springer, 2005.

[57] C. Yang, Y. Li, C. Liu, and X. Yuan. Deep learning-based viewpoint recommendation in volume visualization. *Journal of Visualization*, 22(5):991–1003, 2019.

[58] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[59] L. Zhou and C. Hansen. Transfer function design based on user selected samples for intuitive multivariate volume exploration. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 73–80. IEEE, 2013.