Technische Universität München
TUM School of Computation, Information and Technology

Deep Learning for Volume Visualization

Sebastian Klaus Weiß

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. rer. nat. Matthias Niessner
Prüfer der Dissertation:	1. Prof. Dr. rer. nat. Rüdiger Westermann
	2. Prof. Dr. rer. nat. Dr. techn. h. c. Dr.-Ing. E. h. Thomas Ertl, Universität Stuttgart

Die Dissertation wurde am 20.05.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.09.2022 angenommen.

For peace

Abstract

Visualizing volumetric data is required when inspecting CT and MRI scans in medical imaging, aerodynamic and fluid mechanical simulations, or reconstructions of extraterrestrial events in astrophysics. Applications in these areas produce datasets that exceed thousands of discretization units per spatial axis (voxels) and hundreds of time steps. At those dataset sizes, the time to process or visualize the dataset, i.e. rendering it into a human-understandable image, becomes increasingly expensive. For example, the performance of volume rendering using ray marching, without additional optimizations, scales linearly with the number of pixels on the screen and the number of voxels to traverse. Therefore, optimized rendering strategies for interactive explorations have to be developed. Besides rendering existing datasets, the equally important inverse problem includes the reconstruction of the 3D volume from images, e.g. tomographic reconstruction, or finding camera positions that allow for the optimal inspection of features in the dataset. Those inverse problems require a differentiable rendering routine that can deal with the large dataset sizes mentioned above.

In this dissertation, we first explore acceleration techniques for volume rendering using deep neural networks. We show that the rendering performance of volume visualization can be improved by first rendering a low-resolution image of the dataset, followed by a neural network that upscales this image to a high-resolution version. The neural network can accurately reconstruct the appearance of the high-resolution volume rendering from the low-resolution image using a fully-convolutional network architecture. Temporal stability is achieved via a loss function measuring the difference to the previous prediction, warped by the optical flow, during training. Additionally, ambient occlusion can be estimated during upscaling to improve depth perception. Our method leads to a significant speedup compared to baseline rendering, especially if ambient occlusion is included.

We next investigate whether a network can learn to reduce the sample count even further by sampling the volume only in those regions that carry the most information. A first neural network is tasked to select pixels in the image where the volume is sampled. A second network then reconstructs the final image from those sparse samples. Both networks are trained end-to-end so that samples are

placed where the reconstruction requires the information most. This way, an accurate reconstruction is achieved with only 5%-10% of the pixels on the screen actually rendered.

To make a step from networks acting on images to networks on 3D scene properties, the rendering process itself has to be made differentiable. Previous methods for differentiation include the so-called Forward Differentiation and Adjoint Differentiation. In Forward Differentiation, on one hand, the computation cost scales linearly with the number of parameters to differentiate. Therefore, this approach is unsuitable for tasks that include many degrees of freedom, e.g. volume density reconstruction. Adjoint Differentiation, on the other hand, allows to easily differentiate multiple parameters at once but requires intermediate memory for every basic rendering operation. This approach, therefore, does not scale well with larger datasets, where more voxels per pixel have to be traversed. We show that this restriction can be lifted by analytically inverting parts of the rendering equation. This leads to an algorithm in constant memory, i.e. independent of the number of operations per pixel. We demonstrate possible applications for volume reconstruction and transfer function reconstruction from images, as well as best viewpoint selection. In ongoing projects, we apply this inversion technique to train scene representation networks – a compact grid-free representation of volumes – directly from observed images.

All approaches mentioned above use ray marching to render images of volumes. Ray Marching, however, can lead to rendering artifacts due to possibly unbounded errors in the numerical quadrature of the underlying integrals. Prior work on volume rendering with a controlled error bound apply a transfer function mapping prior to interpolation (pre-shading). This limits the possible level of detail to the grid resolution. We extend upon that and show, that error bounds can also be given when a piecewise-polynomial transfer function mapping is included in post-shaded volume rendering. This allows for renderings with higher-frequency features as well as the inclusion of, e.g., lighting and shading. Potential applications include situations, where an upper limit on the rendering error is required, as well as situations with very narrow and sharp features that would lead to rendering artifacts due to discretization errors in ray marching.

Zusammenfassung

Visualisierungen volumetrischer Daten werden bei der Inspektion von CT und MRI-Scans, in der Strömungslehre bei Fluidsimulationen, oder bei der Rekonstruktion extraterrestrischer Ereignisse in der Astronomie verwendet. Anwendungen in diesen Bereichen liefern Datensätze, die tausende Diskretisierungseinheiten pro Raumachse (Voxel) und hunderte Zeitschritte umspannen. Bei diesen Größenordnungen werden Operationen auf den Datensätzen, wie zum Beispiel die Visualisierung als menschenverständliche Bilder, zunehmend teuer. Beispielsweise skaliert die Berechnungszeit von Volume Rendering mittels Ray Marching, ohne zusätzliche Optimierungen, linear in der Anzahl der Pixel des Bildschirms und der Anzahl der zu traversierenden Voxel. Für interaktive Anwendungen müssen deshalb Optimierungsstrategien entwickelt werden. Neben dem Rendering existierender Datensätze ist das inverse Problem gleich bedeutsam. Inverse Probleme beinhalten die Rekonstruktion von 3D-Volumen aus Bildern, zum Beispiel tomographische Rekonstruktionen, oder das Optimieren von Kameraparametern für eine ideale Perspektive zur Betrachtung der Merkmale im Datensatz. Diese inversen Probleme benötigen eine differenzierbare Rendering-Routine, die mit den oben beschriebenen großen Datenmengen umgehen kann.

In dieser Dissertation erforschen wir Beschleunigungstechniken für Volumenrendering mittels Deep Neural Networks. Wir zeigen, dass die Renderzeiten von Volumenvisualisierungen verbessert werden können, indem zunächst ein niedrig aufgelöstes Bild des Datensatzes gerendert wird, welches dann von einem neuronalen Netz in eine hochaufgelöste Version übersetzt wird. Das neuronale Netz kann das Erscheinungsbild der hochaufgelösten Visualisierung von dem niedrig aufgelösten Bild mittels einer fully-convolutional Netzwerkarchitektur akkurat rekonstruieren. Zeitliche Stabilität wird dabei durch eine Kostenfunktion erreicht, die den Unterschied zum vorherigen Bild, unter Beachtung des optischen Flusses, vergleicht. Zusätzlich kann Ambient Occlusion bei der Hochskalierung durch das Netzwerk mitgeschätzt werden, um den Tiefeneindruck zu verbessern. Unsere Methode führt zu einer signifikanten Beschleunigung des Rendering, insbesondere, wenn Ambient Occlusion eingebunden wird.

Als nächstes analysieren wir, ob ein Netzwerk die Anzahl an Stichproben in das Volumen weiter reduzieren kann, indem die Proben nur in Regionen genommen werden, die die meiste Information tragen. Ein erstes Netzwerk selektiert dazu die Pixel im Bild, an dem das Volumen abgetastet werden soll. Ein zweites Netzwerk rekonstruiert dann das finale Bild aus diesen dünn besetzten Stichproben. Beide Netzwerke werden Ende-zu-Ende trainiert, damit die Stichproben an den Stellen gesetzt werden, die für die Rekonstruktion die meiste Information tragen. Mit der Auswahl von nur 5%-10% aller Pixel im Bild wird mit diesem Ansatz bereits eine akkurate Rekonstruktion erreicht.

Um den Schritt von Netzwerken, die auf 2D-Bildern agieren, zu Netzen auf Eigenschaften im 3D-Raum zu tätigen, muss der Darstellungsprozess selbst differenzierbar sein. Frühere Arbeiten für die automatische Berechnung von Ableitungen nutzen die Methoden der Forward Differentiation oder die Adjoint Differentation. Bei der Forward Differentiation skaliert die Berechnungszeit linear mit der Anzahl der abzuleitenden Parameter. Damit ist diese Methode ungeeignet für Anwendungen mit vielen Freiheitsgraden, wie zum Beispiel die Volumenrekonstruktion. Im Gegensatz dazu erlaubt die Adjoint Differentiation die Ableitung beliebig vieler Parameter auf einmal, jedoch wird zusätzlicher temporärer Speicher bei jeder primitiven Operation benötigt. Dies führt zu Skalierungsproblemen bei großen Datensätzen mit Tausenden zu traversierenden Voxeln. Wir zeigen, dass diese Speicherlimitierung aufgehoben werden kann, indem ein Teil der Strahltransportgleichung analytisch invertiert wird. Das führt zu einem Algorithmus mit einem Speicherverbrauch, der unabhängig in der Anzahl an Operationen pro Pixel ist. Wir demonstrieren als mögliche Anwendungsgebiete die Rekonstruktion von Dichtevolumen und Transferfunktionen aus Bildern sowie optimale Blickpunktselektierung. In laufenden Projekten verwenden wir diese Invertierungstechnik zum Trainieren von Scene Representation Networks, einer kompakten, gitterlosen Repräsentation von 3D-Volumen, aus Bildern.

Alle zuvor genannten Ansätze nutzen Ray Marching zur Berechnung der Bilder. Ray Marching kann allerdings zu Artefakten in den Bildern führen, da die benutze Quadraturmethode der zugrundeliegenden Integrale beliebig große Fehler aufweisen kann. Frühere Arbeiten zu Fehlerabschätzungen im Volume Rendering wenden die Transferfunktion typischerweise vor der Interpolation an (pre-shading). Dadurch wird der mögliche Detailgrad durch die Gitterauflösung begrenzt. Wir erweitern frühere Arbeiten und zeigen, dass Fehlerabschätzungen auch berechnet werden können, wenn eine stückweise-polynomielle Transferfunktionen nach der Interpolation (post-shading) angewendet werden. Dies ermöglicht Renderings mit höher aufgelösten Merkmalen und mit lokaler Lichtberechnung. Mögliche Anwendungen umfassen Situationen, in denen eine obere Grenze für den möglichen Fehler benötigt wird, oder Transferfunktionen mit sehr schmalen, scharfen Spitzen, die zu großen Fehlern bei der Diskretisierung in traditionellem Ray Marching führen.

Acknowledgments

First and foremost, I would like to thank Prof. Dr. Rüdiger Westermann for his supervision and support during my Ph.D. time. I remember uncountable discussions about current research topics, completely different topics and ideas, and just fun remarks about everything. He challenged me to reflect on my own thoughts and broadened the scope of my knowledge. He provided me with guidance when I was stuck, but never restricted me in the research topics I pursued.

I sincerely thank my co-authors, Mengyu Chu, Daniel Cremers, Florian Bayer, Justus Thies, Mustafa Işık, Nils Thuerey, Philipp Hermüller, and Robert Maier for their timely and valuable contributions, even during the last minutes before a deadline. Without them, the papers included in this thesis would not be possible.

Christian Reinbold was the best office mate and friend I could hope for. Over the last years, he always had an open ear for me if I got stuck with a research question. So many ideas were generated and blockages were resolved just by quickly chatting with each other. Besides common research interests, we shared a passion for music. Many a conversation drifted towards discussing recent concerts that we attended or even played ourselves, or new music that we explored. I will miss all the fun we had, inside the office as well as outside in our leisure time.

I also thank Susanne Weitz for helping me with all matters of organization, may it be contracts, business trip applications, or even help with the accounting of new purchases for the university big band I play with. Sebastian Wohner, our system administrator, was always ready to help me with any matters of software, website management, or the installation of a new hard disk if I ran out of available disk space. I also never forget all the jokes about our shared first name and initials.

My colleges Fatemeh, Mengyu, Behdad, Kevin, Ludwic, Björn, Christoph, Christian, Junpeng, Philipp, Marie-Lena, Michael, Alexander, Lukas, and Patrick made the time here at the chair unforgettable. I especially want to thank Junpeng for providing me with various green and black teas.

Lastly, my friends and family always provided me with support throughout my time as a Ph.D. and gave me stability, and sometimes necessary distractions during troublesome times. Thank you.

Contents

1 Introduction

The topic of Volume Visualization coarsely includes methods to generate 2D images from 3D volumes. These 3D volumes arise in various fields, making volume visualization an umbrella term for various methods with interdisciplinary applications. In medical imaging, 3D volumes are obtained from, e.g., CT or MRI scans. Experts are interested in locating anomalies for diagnosing or for planning operations. The former requires a rendering system that allows for an interactive and accurate exploration of the dataset. The latter requires a simulation of the three-dimensional material. In fluid mechanics, the behavior of gasses or liquids like air or water are simulated to investigate the heat distribution or turbulence patterns. Examples include the visualization of the turbulence at airplane wings to optimize the lift and drag or the circulation rolls that occur when hot and cool liquids are mixed [Fra+19]. Such simulations can produce terrabytes of data. For example, the Channel Flow simulation [DMG20] reaches a resolutions of $2048 \times 512 \times 1536$ discretization units (*voxels*) in the three spatial dimensions and is simulated for 4000 steps in time. It requires 23.4TB of uncompressed storage. For those terra-scale datasets, both the storage and the rendering become challenging and require fast algorithms.

To display the 3D volumes, the two most common algorithms are isosurface rendering and direct volume rendering. In isosurface rendering, a single surface is visualized that depicts the locations, where a scalar value like the density takes on a fixed, user-defined value. Rendering methods include raytracing of this surface [Lev88; Lev90a] or discretizations into triangle meshes [LC87], detailed in Sec. 3.1.1. In direct volume rendering, each position in space is mapped to a semi-transparent particle that absorbs a fraction of the light passing through that location and emits light on its own [Max95]. This mapping from the value stored in the volume (typically called the *density*) to absorption (scalar) and emission (red-green-blue) is described by the so-called *transfer function* (TF). The result can then be compared to a colored, self-emitting cloud. There are two possibilities on how this mapping is applied. First, in pre-classification, the TF mapping is applied on the vertices of the grid that store the

values, giving rise to a color volume that is then interpolated. This, however, limits the features that are introduced by the TF to the grid resolution. Second, the TF is applied after the density values are interpolated. This allows the TF to introduce much finer details not limited to the grid resolution. The mathematical details are presented in Sec. 3.1.2.

Direct volume rendering is usually computed via raytracing [Max95]. In the real world, photons are emitted by a light source and sent into the scene. These photons then interact with the objects in the scene and are either absorbed, reflected or refracted, until they reach the eye where the receptors perceive the photons and send the information further to the brain. In raytracing, this process is inverted. Starting from a virtual camera center, rays are sent into the scene through the pixels of the screen. Each ray then interacts with the objects until they reach a light source. In other words, in raytracing, the path of photons is traced in reverse direction. In DVR, this process is simplified by discarding scattering events. Only rays leaving the camera in a straight line are considered. This is called the *emission-absorption model* [Max95]. Still, this algorithm is computationally intense, leading to the development of numerous acceleration schemes, see Sec. 2.1.

In recent years, artificial neural networks have seen great advances and are now applied in all kinds of applications. These include, for example, image classification and segmentation to aid in autonomous driving [JZA18; Gho+19], attention networks for natural language translation and understanding [Vas+17; Dev+18], autoencoders for data compression [The+17; Che+18], recurrent networks to predict future events in weather forecasts [Rei+19] and stock prices [KT90; Sil+17], or denoising [Cha+17] and style transfer networks [RAHK22] to enhance the realism of computer-generated images in games and movies. We want to especially highlight super-resolution networks [Don+16; SSH17]: Given a low-resolution input image, networks are tasked to predict the high-resolution version. In supervised training, the networks are trained on given pairs of low-resolution and high-resolution images. Once trained, the networks can estimate the high-resolution images from novel images, never observed during training. It can be argued from a signal theoretical point of view, that new, semantically plausible features that go beyond of what classical linear or cubic filters can achieve, are impossible. However, once certain assumptions about the domain are made, i.e., what data to expect, neural networks have proven to be efficient in encoding this prior knowledge in their predictions. For example, if the input data is guaranteed to consist of low-resolution photographs of human faces, which, for humans, look like a random collection of a few pixels, neural networks can successfully predict high-resolution images of faces that plausibly explain the low-resolution versions [Hsu+19; Jia+21]. Similar results have been found for super-resolution of text for optical character recognition (OCR) applications [LJ18; XYL20].

Since those networks seem to detect common structures in the data and utilize those during the prediction, it is natural to ask, whether networks can be applied for scientific volume visualizations as well. If the data stems from CT scans of the human head, the networks can assume, that they will

only see CT scans of the human head [You+19]. If the data stems from weather forecasts, certain patterns are prone to repeat themselves and can be exploited by super-resolution networks [Höh+20]. Combining this observation with the fact presented above, that volume visualization is expensive, can neural networks aid in improving the performance of a visualization pipeline by rendering only a low-resolution image and perform super-resolution to reconstruct the high-resolution image?

The super-resolution networks mentioned above take a low-resolution image on a regular grid, the pixel raster, as input. This equates to a sparse sampling of the high resolution image in regular intervals, i.e., only at every 4^{th} pixel in every dimensions, and to reconstructing the dense high-resolution image using a neural network. The underlying assumption behind such methods is thus, that every pixel is equally important. But does this hold true? Are there better ways to select the pixels, i.e. the samples, especially adaptively to the data? For classification tasks, humans on one hand tend to identify the object correctly if just the silhouette or edges are given, whereas traditionally trained networks fail in those cases. Networks, on the other hand, can predict objects more accurately than humans, if just textures are given without shape cues [Gei+19]. This leads to the implication, that neural networks require different inputs to what humans require and expect. Translated to the selection of input pixels for super-resolution, many heuristics have been proposed that adaptively place more samples in more important regions based on human-designed criteria [Tur+19]. These methods assume specific interpolation and reconstruction schemes and do not consider, that, if applied together with networks, different inputs might be more suitable. This leads to the following question: Is it possible to learn, what samples in the input actually carry the most information for a reconstruction network? Can a second neural network learn to predict, where the reconstruction needs samples without having access to all samples?

The downside of neural networks, however, is their inherent errors. The networks are trained to match the distribution of the training dataset. In theory, there always exist neural networks that can approximate a continuous function to an arbitrarily chosen error, called the *universal approximation theorem* [HSW89; Cyb89]. While it is possible to give a constructive proof of the above theorem [CCL92], for practical applications where the size of the network is limited, the networks remains an approximation of the data only. Furthermore, no upper bounds on the errors of the prediction can be computed in most cases. Mathematically proving robustness properties of networks or possible prediction errors remains a challenging topic [YR19]. For the case of image classification, *adversarial examples* are an extreme case of those unbounded prediction errors. Here, small perturbations to the input images that are not noticeable to a human, are constructed in such a way, that the network misclassifies the input with a high certainty [Sze+14; Ily+19; Liu+19a]. Making networks robust against such attacks is a challenging task [WK18]. Therefore, what are applications of neural networks, where errors are acceptable and can be tolerated?

Errors in the resulting rendered images do not only stem from networks. The rendering process itself introduces approximation errors. While analytical solutions for isosurface rendering can be given [LC96; Par+98; Neu+02; Mar+04], for direct volume rendering in the general case, i.e. a grid with an arbitrary interpolation function and an arbitrary TF, no analytical solutions exist [Max95]. For special cases like pre-classification (see Sec. 3.1.2), quadrature schemes with a controlled error bound were presented by Novins and Arvo [NA92]. This leads to the question, is it possible to derive error bounds and a high-precision quadrature scheme for the post-classification case?

As the last challenge, we investigate differentiable volume rendering. So far, only the image formation process was investigated. This means, the volume data, the camera parameters and the transfer function is given and only the resulting image has to be generated. However, when some of those parameters are unknown, they have to be reconstructed from images. This includes optimizations such as best-viewpoint-selection or tomographic reconstruction, where the volume is reconstructed from X-ray images or other detection systems. Efficient optimizers like Adam [KB15] are gradient-based optimization schemes, i.e., they require the gradients of the image with respect to the parameter to optimize. Differentiable rendering frameworks like Mitsuba 2 [ND+19; ND+20] allow for the computation of such derivatives, but are memory and/or computational intensive. How can direct volume rendering be differentiated in a memory and computational efficient way?

1.1 Contribution

In collaboration with other researchers, several contributions are presented in this thesis that address the research questions mentioned above. We show [Wei+21], that super-resolution networks can improve the rendering performance of large datasets for isosurfaces. To compensate for prediction errors, the networks are embedded in an interactive exploration framework with foveated rendering [Gue+12]. In foveated rendering, the area around the focus region of the eye (the fovea) is rendered with high resolution and the resolution falls off towards the periphery. Here, the focus region is rendered accurately and the network predictions are used in the periphery. As an extension of the previous work, we present an adaptive sampling pipeline that features an importance network tasked with estimating the sample locations with the most information for the following reconstruction step [Wei+20]. The two approaches above both apply networks on images of isosurface or direct volume renderings. Hence, they incorporate errors from the neural network, as well as from the renderer itself. To provide a baseline where errors in the rendering are minimized or at least known to not exceed certain error bounds, we present controlled-precision direct volume rendering for a larger class of TFs than previously possible [WW21]. To go beyond images and infer features of the underlying 3D data, a differentiable rendering pipeline is needed. We present two algorithms [WW22], how direct volume rendering can be differentiated with bounded memory to allow for an arbitrary number of steps per

ray and select the best-performing algorithm per use case. The following list summarizes the concrete contributions of this thesis:

- Direct volume rendering typically relies on numerical quadrature to estimate the volume rendering integral as analytical solutions are not possible in the general case. There exist analytical solutions if the data is not given on a hexahedral grid (a voxel grid), but instead on a tetrahedral grid. In tetrahedra, barycentric interpolation leads to a linear function of the density. This allows for an analytical solution of the volume rendering integral [WM92], exploited, e.g. for transfer functions (TFs) based on sums-of-gaussians [Kni+03] or for pre-integrated TFs [EKE01]. We present a quadrature scheme [WW21] that, for a specific class of TFs, specifically piecewise polynomial TFs, allows controllable error bounds also for the post-classification case where the TF is applied after tri-linearly interpolating the density values. First, we apply accurate isosurface intersection tests [LC96; Par+98; Neu+02; Mar+04] on the control points of the piecewise polynomial TF, combined with a voxel traversal [AW87; JTC14]. This leads to a segmentation of the ray with a polynomial function of the density, absorption, and color per segment. Then, we apply the quadrature scheme by Novins and Arvo [NA92], originally introduced for pre-classification, to solve the volume rendering integral to high precision. This allows to use TFs with very sharp and narrow peaks and greatly reduces the errors in those cases if compared to traditional quadrature schemes using a constant step size.
- We present a super-resolution method for isosurface renderings that takes low-resolution renderings of the object and estimates high-resolution images with $4\times$ the number of pixels per dimension [Wei+21]. The architecture is based on the EnhanceNet architecture by Sajjadi *et al.* [SSH17]. The original architecture works on RGB-images. We show that a better quality of the predicted high-resolution isosurface renderings is achieved by upscaling the normal map instead of the color images. This additionally decouples the shading from the network and allows to change the color of the surface or the light position afterward. Furthermore, the network is tasked with predicting global illumination in the form of ambient occlusion (AO). AO approximates the shadowing in cavities as less reflected light reaches those cavities. Including AO greatly enhances the depth perception of the object. While fast approximations of AO in screen-space exist [Mit07; SA07; BS08], computing the true shadowing value requires tracing many secondary rays in 3D which is costly. We show that the super-resolution network can jointly learn to estimate AO together with the high-resolution representation of the object, and include an application in the context of foveated rendering [Gue+12].
- In the method presented above, every pixel was treated equally important. It can be interpreted as a sub-sampling, where only every $4 * 4 = 16^{\text{th}}$ pixel is rendered, arranged in a regular grid, and the network fills in the missing values (the reconstruction). We hypothesize, that there

exists a better placement of those rendered pixels that supplements the reconstruction network with the information needed. We show, that a neural network can be trained to automatically distinguish important and redundant areas for a second reconstruction network [Wei+20]. This first network called the importance network is not trained directly, i.e., by explicitly enforcing important regions, instead it is trained indirectly through the second reconstruction network. To enable this indirect training, we propose a differentiable sampling stage that allows the propagation of derivatives from the reconstruction network to the importance network. The proposed method allows to render images with a user-defined budget of rays to render and outperforms fixed super-resolution methods as above in terms of quality for the same number of samples.

- The two methods presented above utilize networks that act on images. To infer information about the underlying 3D structures themselves from images, a differentiable rendering algorithm is needed. While such algorithms exist, they are either memory-limited [ND+19] or only support derivatives for a subset of the parameters, e.g., the volume densities [van+15; Aar+16] or material properties [ND+20]. Mitsuba 2 [ND+19], for example, uses automatic differentiation using Adjoint Differentiation to compute the gradients and requires storing every intermediate result along the ray. We show that for the case of direct volume rendering, storing intermediate values can be avoided [WW22]. The main observation is, that the blending step that takes the previous blended color and the current contribution and computes the updated color, can be inverted. By recomputing the current contribution, the previously blended color can be recomputed and does not have to be stored. This allows computing derivatives for renderings with an arbitrary number of steps along the ray with a constant memory cost. We demonstrate the capabilities of the differentiable rendering pipeline for camera location, transfer function, and volume density optimizations. In unpublished work [WHW21], this technique is also applied to train networks representing the 3D volume from images. We further show, that for scenarios with a low number of parameters, e.g. the optimization of the camera extrinsics, a different differentiation scheme, so-called Forward Differentiation, can compute the derivatives faster than Adjoint Differentiation.

1.2 Outline

The following sections of this thesis are structured as follows. In Chapter 2, related work regarding the research areas related to the contributions of this thesis is discussed. This includes traditional, non-network-based acceleration structures, an overview over recent advances in super-resolution networks, the application of networks to volume visualization, differentiable rendering, and controlled-precision rendering. The fundamentals and the core ideas of the contributions of this thesis are presented in

Chapter 3. The published papers that are part of this thesis are summarized starting in Chapter 4, together with the contributions of the individual authors. Future work and a general summary are provided in Chapter 8. The published papers associated with this thesis are appended at the end of this document.

1.3 List of Publications

The methods described in this thesis have been originally proposed and published in the following peer-reviewed journal articles and conference proceedings:

- Sebastian Weiss, Mengyu Chu, Nils Thuerey, and Rüdiger Westermann.
  "Volumetric Isosurface Rendering with Deep Learning-Based Super-Resolution".
  In: *IEEE Transactions on Visualization and Computer Graphics*, Volume 27, Issue 6, June 2021, pp. 3064 – 3078.
  doi:10.1109/TVCG.2019.2956697
- Sebastian Weiss, Mustafa Işık, Justus Thies, and Rüdiger Westermann.
  "Learning Adaptive Sampling and Reconstruction for Volume Visualization".
  In: *IEEE Transactions on Visualization and Computer Graphics*, November 2020, early access.
  doi:10.1109/TVCG.2020.3039340
- Sebastian Weiss and Rüdiger Westermann.
  "Differentiable Direct Volume Rendering".
  In: *IEEE Transactions on Visualization and Computer Graphics*, Volume 28, Issue 1, January 2022, pp. 562 – 572.
  doi:10.1109/TVCG.2021.3114769

The following publication has been described in this thesis but is not relevant for examination:

- Sebastian Weiss and Rüdiger Westermann.
  "Analytic Ray Splitting for Controlled Precision DVR".
  In: *EuroVis 2021 - Short Papers*, The Eurographics Association, 2021. **Short Paper**
  doi:10.2312/evs.20211051

Further publications that are not part of this thesis include:

- Sebastian Weiss, Jun Han, Chaoli Wang, and Rüdiger Westermann.
  "Deep Learning-Based Upscaling for In Situ Volume Visualization".

In: Hank Childs, Janine C. Bennett, and Christoph Garth (eds), *In Situ Visualization for Computational Science*, 2022, pp. 331 – 352.
doi:10.1007/978-3-030-81627-8_15

- Sebastian Weiss, Philipp Hermüller, and Rüdiger Westermann.
"Fast Neural Representations for Direct Volume Rendering".
ArXiv-preprint 2021, **not peer-reviewed**.
doi:10.48550/arXiv.2112.01579

- Sebastian Weiss, Robert Maier, Daniel Cremers, Rüdiger Westermann, and Nils Thuerey.
"Correspondence-Free Material Reconstruction using Sparse Surface Constraints".
In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4686 – 4695.
doi:10.1109/CVPR42600.2020.00474

- Sebastian Weiss, Florian Bayer, Rüdiger Westermann.
"Triplanar Displacement Mapping for Terrain Rendering".
In: *Eurographics 2020 - Short Papers*, The Eurographics Association, 2020.
doi:10.2312/egs.20201016

- Alex Frasson, Martin Ender, Sebastian Weiss, Mathias Kanzler, Amrish Pandey, Joerg Schumacher, Rüdiger Westermann.
"Visual Exploration of Circulation Rolls in Convective Heat Flows".
In: *IEEE Pacific Visualization Symposium (PacificVis)*, 2019.
doi:10.1109/PacificVis.2019.00031

2

Related Work

In this chapter, we discuss prior work in volume visualization and related topics. The ideas presented here form the basis of the contributions of this thesis. We start with a discussion of traditional, non-network-based acceleration strategies for rendering volumetric data in Sec. 2.1. These are orthogonal methods that are further enhanced by network-based acceleration strategies, see Paper A and Paper B. The foundations of the methods used in these two publications are image and video super-resolution approaches that are discussed in Sec. 2.2. Further and concurrent work in applying neural networks in volume visualizations are summarized in Sec. 2.3. All these methods above have the potential to introduce errors in the image. To analyze, how a baseline with a low error can be achieved, controlled-precision volume rendering is discussed in Sec. 2.5, later applied in Paper C. To investigate features in the data itself from image observations, see Paper D, a differentiable rendering pipeline is needed. Related work in that regard are summarized in Sec. 2.4.

2.1 Non-network-based Acceleration Strategies for Volume Rendering

Volume rendering has seen a first use in 1983 in medical imaging where the 3D structures of CT or MRI scans are visualized [UH83; Rey+85]. Since then, there has been a stride for even larger datasets with more fine details while improving the rendering performance to allow for interactive visualizations. In this section, we explore acceleration techniques for volume rendering that make use of "traditional" algorithms and datastructures without neural networks. Neural networks as presented in Sec. 2.3 often built upon those optimizations and provide additional, orthogonal improvements.

Acceleration of Isosurface Rendering As briefly mentioned in Chapter 1, one way to render isosurfaces is by walking along the ray shot from the camera through each pixel [Lev88; Lev90a]. Typical datasets exhibit large empty regions that do not need to be traversed. This naturally leads to

employing tree data structures [SW88]. In such data structures, the leaves partition the available space and parent nodes contain aggregate statistics of the child nodes. A specific case is the *octree*, where each node has exactly 8 children that partition the area of the parent node in a regular $2 \times 2 \times 2$ grid. If the isosurface is fixed, each voxel can be classified as either "empty" or "not empty". Levoy [Lev90a] shows how to build a static octree that encodes this binary classification. During rendering, instead of traversing the volume using a constant step size, the nodes in the octree are queried and the node and its children are only recursively traversed if they contain at least one voxel classified as "not empty". Further research in that direction includes the works by Knoll *et al.* [Kno+06; KWH09] on efficient memory layouts of the octree and traversal algorithms. Independent of isosurfaces or volume visualization, such sparse tree structures have seen great success in rendering general voxelized geometry [LK10; KSA13; Dad+16; CBE20] or accelerating intersection tests for raytracing of opaque scenes [Gun+07; Hap+11] and is now even implemented in hardware since the NVIDIA Turing architecture [NVI18].

If the isovalue is fixed, the volume can also be transferred into a signed distance field (SDF) representation [CS94; JS01]. The SDF stores at every location the Euclidean distance to the nearest surface, with a sign specifying if the position is inside (negative) or outside (positive) of the object. Regardless of the ray direction, the SDF value at the current location represents a lower bound on the step size where it is guaranteed, that no intersections are missed. This idea of an adaptive step size is formalized in the sphere tracing algorithm, introduced by Hart *et al.* [HSK89; Har96]. Due to the cost of constructing the SDF, this method has not seen much attention compared to other acceleration structures, e.g. the tree structure described above. However, SDFs have resurfaced in recent years for volumetric reconstruction tasks in computer vision [New+11; Iza+11; Per+15], based on ideas dating back to the work by Curless and Levoy [CL96].

Acceleration of Direct Volume Rendering For direct volume rendering (DVR), the ray does not terminate at the first intersection with an opaque surface, but instead accumulates emission terms along the whole ray. While the acceleration technique presented by Levoy can still be used for DVR, once a node is classified as "non empty", no further distinction is made. The region covered by that node might be homogeneous and a coarse step size suffices, or it is inhomogeneous and a fine step size is needed. Danskin and Hanrahan [DH92] realize this idea by the introduction of a min-max tree that stores the minimal and maximal density in that region. Danskin and Hanrahan only support volumes without a transfer function (TF) mapping. One of the first methods to incorporate TFs is presented by Meißner *et al.* [Mei+01]. Here, when the TF is changed, a coarse grid – instead of a hierarchical tree – is recomputed with a classification if the area spanned by each coarse cell contains a non-transparent voxel. This coarse grid is then traversed to skip over empty regions. Sobierajski and Kaufman [SK94] approach accelerating the rendering by defining primitive bounding objects that restrict ray-traversals

to a smaller area containing the volume. A similar method using intersection tests against bounding objects is employed by Wan *et al.* [WKB99].

With the introduction of programmable graphics hardware (GPUs), significant performance improvements have been achieved by approaches which exploit high memory bandwidth and texture mapping hardware on GPUs for sampling and interpolation in 3D scalar fields [KW03; Had+05]. Despite the massively parallel architecture of GPUs, some CPU-based raytracers could still outperform GPU implementations [Kno+11]. For isosurface ray-casting, frame to frame depth buffer coherence on the GPU was employed to speed up first-hit determination [Kle+05; Bra+09].

As GPUs are mostly targeted to the efficient rasterization of triangles, several works have employed hybrid schemes between rasterization and raytracing. For isosurfaces, Reichl *et al.* [Rei+12] dynamically switch from rasterization to raytracing in areas where large overdraw in triangle rasterization would lead to degraded performance. More recently, Hadwiger *et al.* [Had+18] introduce SparseLeap, a method employing pyramidal occupancy histograms to generate geometric structures representing non-empty regions. These are rasterized into per-pixel fragment lists to obtain tight segments along the ray that contain the volume. Only these segments are then traversed in a second raytracing step. With the introduction of the NVIDIA's Turing architecture [NVI18], ray traversal through bounding volume hierarchies (BVH) is supported directly in hardware, as mentioned above. This is exploited in several recent works, e.g., by Ganter and Manzke [GM19], Morrical *et al.* [Mor+19], or Wald *et al.* [WZM21]. For a comparison of several bounding volume hierarchies for ray tracing, we refer to the summary article by Meister *et al.* [Mei+21].

Reducing the number of rays All above methods optimize the performance per ray. The computation cost of the rendering still scales linearly with the number of pixels on the screen. Therefore, it is desired to reduce the number of rays and render the image at a lower image resolution. The goal is now to assign an importance value to each pixel so that rendering time is only spent in areas that carry the most information. Early works start with a low-resolution image and render more pixels in areas with the largest variance [PS89; Lev90b; RFS03]. For example, Kratz *et al.* [Kra+11] render two images at two different, coarse resolutions, and refine in areas with large differences between those images. Further works include perceptual models [BM98; Mys98; RPG99], image difference operations [LDC06], or entropy-based measures [Xu+05]. A special case is foveated rendering, where the properties of the human eye are exploited [Gue+12]: Humans have densely packed photoreceptors only in a small area, a circle of 5° around the gaze regions. Outside of this small area, the observable spatial resolution quickly falls off. This is exploited in several recent works to reduce the image resolution in the off-focus regions, including Tursun *et al.* [Tur+19] to incorporate luminance information, Bruder *et al.* [Bru+19] for an efficient sampling and interpolation scheme, Kim *et al.* [Kim+19] in an application with a head-mounted display and eye tracking, or Frieß *et al.* [Fri+20] to utilize different quality settings

for a H.264 encoder in different regions. Recent works by Kaplanyan *et al.* [Kap+19] also utilize neural networks for foveated rendering. First, a sparse sampling of the input is performed, where more samples are rendered in the fovea and less samples in the periphery. Then, a neural network interpolates these samples and estimates missing details. In this work, the sampling pattern moves with the gaze position of the eye, but is otherwise fixed. We show, how a neural network can predict the sampling pattern to adapt to the data [Wei+20]. Further works on deep learning in visualization are summarized in Sec. 2.3. Instead of reducing the number of pixels to render, orthogonal approaches reduce the number of samples along the view rays by adaptively choosing the step size [NA92; DH92; Lin+13; CCF15].

Out-of-core rendering If the dataset is too large to fit into the available video memory for GPU raytracers, *out-of-core* strategies have to be employed. In these strategies, the data is split into chunks and then sent to the GPU for rendering chunk-by-chunk, often combined with fast compression schemes [GMIG08; Tre+12; FSK13; MAG19; DMG20]. For a thorough overview of GPU approaches for large-scale volume rendering, let us refer to the report by Beyer *et al.* [BHP15].

2.2 Image and Video Super-Resolution

Before we discuss how neural networks can improve or augment the volume visualization techniques described in the previous section, we first introduce image and video super-resolution methods as they form the basis of many network architectures used later on.

Image Super-Resolution The task of image super-resolution is posed as following: given a low-resolution image, the neural network then predicts the high-resolution image matching the low-resolution version. The networks are trained using pairs of low-resolution images and known high-resolution images as ground truth. For a detailed introduction to the mathematical methods for super-resolution networks, we refer to Sec. 3.3.2. The first work that applies neural networks with a quality surpassing traditional techniques in terms of peak signal-to-noise ratio (PSNR) is presented by Dong *et al.* [Don+16]. From there on, deeper networks are introduced [KKLML16] or Laplacian pyramids utilized to learn features over multiple scales [Lai+17]. In the other direction, Shi *et al.* [Shi+16] propose a sub-pixel convolutional operation in combination with a smaller network to target real-time super-resolution. Further advances include architectures like the ResNet [He+16; Led+17] and DenseNet [Hua+17; Ton+17]. To train the network to predict high-frequent features that cannot be described by traditional image metrics like the PSNR or the structural similarity index (SSIM) [Wan+04], advanced loss functions were developed.

Perceptual losses [Zha+18] send the images to pre-trained networks – networks typically trained for image classification tasks – and compare activations in the inner layers of those networks. The intuition behind those perceptual losses is the observation, that those pre-trained networks for image classification encode semantic information otherwise not captured in simple vector norms like the mean square error. Those features are also called the *content representation* [GEB16]. Training new networks with those perceptual losses forces the network to match semantic features instead of simply matching the ground truth pixel-by-pixel. One of the first works to apply such perceptual losses is the work by Johnson *et al.* [JAFF16] or the EnhanceNet architecture by Sajjadi *et al.* [SSH17].

A step further, in adversarial training (GANs) [Goo+14; ACB17; Gul+17], such a second network is not pre-trained and then fixed, but rather trained jointly with the super-resolution network (the generator). The second network, called the discriminator, is trained to distinguish "true" examples from the training set from "fake" examples from the generator. The generator network is then tasked with "tricking" the discriminator network. Both networks are trained at the same time in a ping-pong fashion. One of the first architecture to employ adversarial training in image super-resolution called SRGAN is presented by Ledig *et al.* [Led+17]. Later works include, e.g., ESRGAN [Wan+18] and SRFeat [Par+18] and have shown an even further increase in image quality. For a further discussion of super-resolution networks for images, we refer to the surveys by Yang *et al.* [Yan+19b] and Wang *et al.* [WCH21].

Video Super-Resolution To extend image super-resolution to videos, the algorithms have to deal with the additional time dimension. Since super-resolution networks are tasked with estimating high-frequency details, a fundamentally ill-posed problem, video super-resolution methods are tasked with ensuring temporal coherence and consistent predictions. Works in this regard can be differentiated whether they use previous low-resolution frames as input [Kap+16; Tao+17; Liu+17; Jo+18] or previous high-resolution estimates [SVB18]. The critical component of video super-resolution is motion compensation and estimation. Most methods first employ a special motion-estimation network [Lia+15; Xie+18; SVB18; Chu+20] that warps the previous input images based on the optical flow. To improve the quality of the warping, Tao *et al.* [Tao+17] propose a sub-pixel warping layer. Jo *et al.* [Jo+18] avoid explicit warping of the previous frame and instead use 3D convolutions directly in the spatial-temporal domain. Li *et al.* [Li+19] later improved such spatial-temporal convolution blocks especially regarding the video quality and computational cost. To deal with large motions over longer time series, Isobe *et al.* [Iso+20] present a hierarchical approach by fusing sequences of different frame rates. With regards to loss functions, Chu *et al.* [Chu+20] employ adversarial training with a discriminator that receives temporal sequences. This combines the advantages of GAN training to produce high spatial detail while retaining temporal coherence. For a further discussion and comparison of recent developments, we refer to the survey by Liu *et al.*[Liu+22]. Such video super-resolution methods are the basis for our proposed super-resolution method for isosurfaces [Wei+21].

A related task is video frame interpolation. Given two frames at time t_1 and time t_2, estimate the frame at an intermediate time $t_1 < t < t_2$. Niklaus *et al.* [NML17a; NML17b; NL18] and Wu *et al.* [WSK18] combine optical flow estimations and neural networks to reconstruct intermediate timesteps. Meyer *et al.* [Mey+18] utilize phase-based motion estimation to better handle large motion. Ideas from those works are then later reused for temporal interpolation of 3D volumetric data, see Sec. 2.3. More recent works apply frame interpolation to speed-up Monte Carlo renderings and utilize auxiliary features like normal maps obtained during rendering [Bri+21].

2.3 Deep Learning in Volume Visualization

With the success of deep learning in various application areas, most noticeably image super-resolution as summarized in Sec. 2.2, neural networks have seen more and more applications in volume visualizations. In this section we give a selection of neural network approaches focussed on volume visualization. For an extended summary of recent advances in deep learning for scientific visualizations including applications in a broader scope, e.g. fluid simulations or particle tracing, we refer to the review article by Wang and Han [WH22].

Super-resolution in space and time For 2D images, You *et al.* [You+19] apply super-resolution (SR) networks with GAN training on slices of CT scans. As there are dozens of SR architectures proposed in the last years, Höhlein [Höh+20] compare multiple architectures for the task of 2D upsampling of wind fields using a qualitative and quantitative study. Those ideas on using 2D SR-networks for visualization are also utilized by the proposed work on image super-resolution for isosurfaces with a focus on temporal stability and global illumination prediction (Paper A, [Wei+21]), or using an adaptive selection and sampling of the pixels for reconstruction (Paper B, [Wei+20]).

The early work by Zhou *et al.* [Zho+17] utilizes three layers of 3D convolutions to upscale a 3D volume directly with a Euclidean Loss against ground truth high-resolution volumes. In parallel, Xie *et al.* [Xie+18] (tempoGAN) show how to perform temporally stable super-resolution for 3D fluid flow via a spatial and a temporal discriminator during GAN-training. Those ideas are improved upon with the SSR-TVD architecture [HW22] for spatial super-resolution of scalar fields and with SSR-VFD [Guo+20] for vector fields. Similar to SSR-VFD, Sahoo and Berger [SB21] perform super-resolution for vector fields, but include streamline integration in the loss function to compensate for accumulated prediction errors over longer time sequences. Han and Wang [HW20] introduce TSR-TVD, an architecture for temporal interpolation of 3D volumes using a recurrent generative network (RGN). The RGN is trained to predict a fixed number of in-between timesteps. To combine both ideas, joint spatial and temporal super-resolution methods were proposed by Han *et al.* [Han+22] (STNet) for scalar fields and by An *et al.* [An+21] (STSRNet) for vector fields.

Feature Prediction and Reconstruction Since networks inherently produce lossy approximations of the data, Tkachev *et al.* [TFE19] embrace this fact and use the difference between the ground truth volume and network predictions as a measure for regions of interest. Those regions are then highlighted for further inspection by experts. Porter *et al.* [Por+19] employ an autoencoder architecture – a network that takes the full-resolution volume, compacts it into a low-dimensional latent-space, and reconstructs the volume from that latent-space – to extract representative time steps from time-series data. Latent-space representations of each time step are projected into a two-dimensional space using t-SNE [MH08] and then selected based on different selection and distance criteria. Berger *et al.* [BLL19] use GANs to model the whole rendering pipeline from camera and transfer function settings to the final image via a neural network. Since the network implicitly encodes the relationship between the TF and the image, it allows to analyze the sensitivity of how changes in the TF influence the resulting image. Similarly, He *et al.* [He+19] train a network on the relationship between the parameters of simulation in an ensemble and the visualization. This allows the exploration of new simulation parameters by directly predicting the expected rendered image for that set of parameters. To analyze the relationship between parameters of multi-variable dataset, Han *et al.* [Han+21] propose a variable-to-variable translation scheme. First, the different variables in the multi-variable volume are clustered based on their similarity in a learned feature space. Then, a neural network translates from one variable to another variable in the same cluster.

Specialized for flow visualization, Han *et al.* [Han+19] show how to reconstruct a 3D flow field from streamlines using deep learning, later improved by Gu *et al.* [Gu+21] for time-dependent flow fields. First, a low-resolution vector field is initialized that matches the given streamlines. This field is then upscaled using a 3D convolutional neural network to obtain the final high-resolution flow field. For selecting representative flow lines, Han *et al.* [HTW20] discretize each flow lines into a binary volume, compute a latent-space representation of that volume using an autoencoder network and use the resulting latent vector as a selection and clustering feature.

Volumetric Compression Recently, neural networks have also been applied to volumetric compression tasks and can exceed traditional compression schemes [BRLP19; DC16; Zha+20] in terms of the achieved compression rate for a given quality. Wurster *et al.* [Wur+21] apply 3D super-resolution networks for compression. The original dataset is iteratively downsampled in an octree-like subdivision as far as possible so that the reconstruction using the network still satisfies a given error bound. Lu *et al.* [Lu+21] utilize scene representation networks, fully-connected networks that learn a direct mapping from a position in space to the scalar value to encode. They achieve high compression rates and allow for random access into the compressed representation. In unpublished work, we show how to accelerate the training and rendering performance of this method with the introduction of a learned latent grid and fast CUDA inference [WHW21].

2.4 Differentiable Rendering

To infer features about the data from images or reconstruct certain properties, the rendering process itself needs to be differentiable. Differentiability allows to optimize the parameters using gradient-based optimizers. In this section we summarize early works and recent developments in differentiable rendering and parameter optimizations.

Differentiability of Opaque Scenes Before differentiable Monte Carlo path tracers were computationally feasible and allow for optimizations "in-the-wild", i.e. from real images, early works make use of special measuring tools. For example, Seitz *et al.* [SMK05] show, that it is possible to decompose the reflections in a scene in images representing the illumination after a single bounce, two bounces, three bounces, and so on. This decomposition is made possible by sending a single laser beam into the scene, moving it across the object, and recording the resulting illumination. This idea was later improved by Nayar *et al.* [Nay+06] using a moving checkerboard pattern instead of a single laser beam. These ideas of deriving information about the scene are then later heavily used in the reconstruction of participating media, see below.

Early works for scenes comprised of opaque objects assume local illumination and smooth shading, i.e. no support for shadows or other global effects. This includes the OpenDR-framework [LB14], smoothing approximations using Gaussian filters [Rho+15], smooth interpolations during the backward pass [KUH18], or using a smooth z-buffer [Pet+19]. Similarly, Liu *et al.* [Liu+19b] use a smooth probability function modeling the probability of a pixel seeing a specific triangle. This method was later efficiently implemented in PyTorch3D [Joh+20]. One of the earliest work to explicitly model the discrete discontinuities at the visibility boundaries was presented by [Li+18] using edge sampling. To avoid self-intersections of the optimized meshes, Nicolet *et al.* [NJJ21] introduce a preconditioner into the gradient descent that favors smooth intermediate results. For an extended summary of the works on differentiable rendering, we refer to the survey by Kato *et al.* [Kat+20].

A completely different approach to achieve a differentiable renderer is using so-called neural rendering. Here, the whole rendering pipeline, from the input objects to the final image, is approximated by a neural network. As an example, Nguyen-Phuoc *et al.* [NP+18] propose "RenderNet", a combination of convolutional and fully-connected networks that replaces the mesh rasterizer. By approximating the rasterizer using differentiable networks, the rendering process becomes implicitly differentiable. The works by Berger *et al.* [BLL19] and He *et al.* [He+19], described in Sec. 2.3, follow the same idea, targeted for volume visualization, but specialize to a specific scene. For a comparison and summary of various approaches in neural rendering, we refer to the review article by Tewari *et al.* [Tew+20]. Recently, the term "neural rendering" is used in a broader sense to also incorporate methods for scene representation for novel view synthesis [TZN19; SZW19; Mil+20] or scene relighting [Bi+20; Sri+21;

Rai+22], that do not specifically target the differentiability of the input object for reconstruction tasks. For a summary of neural rendering in this wider scope, we refer to Tewari *et al.* [Tew+22].

Differentiability of Participating Media For reconstructions of participating media, i.e. volumetric objects like clouds with complex scattering patterns, several reconstruction methods were proposed for controlled laboratory environments. The most prominent parameters of such participating media are the phase function, specifying in which direction a photon is most likely to be scattered, and the absorption and scattering coefficients, specifying the amount of absorption and scattering in the media. Hawkins *et al.* [HED05] show how a time-varying smoke plume can be measured by sweeping a laser beam across the volume and interpreting the resulting images as slices of densities. This reconstruction method assumes that extinction and multiple-scattering effects can be neglected. Similarly, Narasimhan *et al.* [Nar+06] dilute fluid samples of milk, wine, or other fluids until the medium is thin enough to neglect multiple scatterings. The authors use a direct search method, i.e. without gradients, to optimize the absorption and scattering coefficients, as well as the parameter of the Henyey-Greenstein phase function [HG41]. To support multiple scattering events, Mukaigawa *et al.* [MYR10] perform a decomposition into a light field for the direct illumination, one light field after the first bounce, another after the second bounce, and so on. This decomposition is made possible by illuminating the scene using a high-frequency checkerboard pattern [Nay+06]. To keep the computation tractable, they assume a thin volume and approximate it as a 2D object. One of the first methods that fully supports 3D objects and arbitrary phase functions is presented by Gkiolekas *et al.* [Gki+13]. The optimization includes the phase function, scattering and absorption coefficients, but is limited to homogeneous media and relies again on a setup including calibrated lasers.

A special case of volume reconstruction is tomographic reconstruction in medical imaging. X-rays are barely scattered in human bodies and therefore, reconstruction methods can assume a pure absorption model. In this case, the problem becomes a linear problem and specialized algorithms can efficiently reconstruct the volume. Those include algebraic reconstruction methods [GBH70; GB08], filtered backprojection methods [BMB97; MO74; MKH18], or using the Fourier slice theorem [Dev85; Wal00]. The Fourier slice theorem allows for a faster reconstruction, but limits the reconstruction to orthographic projections. For a detailed introduction, we refer to the book by Herman [Her09]. We want to also highlight here the ASTRA-toolbox by van Arle *et al.* [van+15; Aar+16], an easy-to-use algebraic reconstruction toolkit based on an algebraic method called the simultaneous iterative reconstruction technique (SIRT) [GB08]. If other wavelengths are used, scattering events gain influence again, opening up the field of Diffuse Optical Tomography. One example application is tissue reconstruction using near-infrared light for breast cancer detection [HSM00]. We refer to the articles by Boas *et al.* [Boa+01] and Gibson *et al.* [GHA05] for an introduction and a summary of this topic.

The ideas of tomographic reconstruction were then later translated to the field of computer vision and simulations, e.g. by Ihrke and Magnor [IM04] on reconstructions of flames. This method was then later accelerated and extended for the reconstruction of smoke plumes [Gre+12; Oka+15]. Recently, Eckert *et al.* [EHT18; EUT19] show how to couple volume reconstructions with a fluid simulation to reconstruct a time-dependent density and velocity field from a single or a few video streams of a real-world smoke plume.

General-Purpose Differentiable Rendering Algorithms To the best of our knowledge, one of the first works that support differentiable rendering with any kinds of media – opaque or transparent –, various materials, textures, and cameras in a unified framework is Mitsuba 2 by Nimier-David *et al.* [ND+19]. The authors utilize Adjoint Differentiation, see Sec. 3.2.2 to propagate gradients through the various modules of the renderer. This allows to compute gradients for any number of parameters, but involves a high memory cost, limiting the number of rays that can be traced simultaneously. In a follow-up work, Nimier-David *et al.* [ND+20] address this issue in the method called *radiative backpropagation*. This method allows for optimization in constant memory, but increases the computation time to being quadratically in the number of scattering events. We present in Paper D [WW22] a different method to reduce the memory requirements, that also exhibits constant memory requirements, but only a linear computation time in the number of events. The idea is to invert parts of the rendering algorithm to reconstruct intermediate variables that, thus, do not need to be stored anymore. This method, however, is specialized to direct volume rendering without multiple scattering.

Based on Mitsuba 2, several works have further progressed the performance of the differentiable renderer or included more advanced rendering effects. A similar method to reduce the computation cost as in Weiss and Westermann [WW22] by analytically inverting parts of the rendering process is presented by Vicini *et al.* [VSJ21] for path tracing. Nimier-David *et al.* [ND+21] extend Mitsuba 2 with a texture-space optimization scheme to untangle lighting information from material properties for scene editing and relighting, and later with a differentiable ratio tracking to reduce noise and bias in the derivatives of volumetric densities [ND+22].

Parameter Optimization in Volume Visualization For volume visualization, more specifically direct volume rendering, several other methods were proposed that allow for the optimization of specific parameters. One application is the selection of an optimal camera position and orientation to best show the features of the object. The critical part is the definition of what constitutes a “good” viewpoint. One method to measure the quality of a viewpoint is using image entropy as shown by Vázquez *et al.* [Váz+01]. This idea was later applied to volume visualization by Bordoloi and Shen [BS05] and Ji and Shen [JS06]. In the latter work, the authors also show how to extent the idea

to time-varying views where a smooth camera path should be generated. Vázquez *et al.* [VMN08] generate a camera path visiting all interesting viewpoints for a static model. Further work by Mühler *et al.* [Müh+07] include additional terms in the image quality metric to consider different important objects in a segmented model. Similarly, Tao *et al.* [Tao+09] define two entropy measures for the overall shape and detailed features to allow for greater control by the user. For a more theoretical overview of entropy and information theory in visualization, we refer to Chen and Jänicke [CJ10].

Instead of using entropy measures, other works have considered importance measures of specific features like isosurfaces [Tak+05] or feature clustering approaches [ZAM11]. Tao *et al.* [Tao+16] use a voting system where images of volumes from published articles were collected and used as a representation of what constitutes a "good" viewpoint. Then, using a set of feature descriptors, these reference images cast votes to their most similar images in a set of sampled viewpoints and the view with the most votes is selected. Recently, convolutional neural networks have been used to train such feature descriptors in a data-driven way [ST19; Yan+19a].

In the methods listed above, the best viewpoint was selected by sampling many views around the object, e.g. via the Fibonacci sphere algorithm [Mar+13], and then select the best view based on the defined image measure. As an acceleration, Zhang and Wang [ZW10] use the shuffled frog leaping algorithm to converge to a local optimum more quickly. We show in Paper D, how a differentiable renderer allows gradient-based optimization for a fast convergence against a locally optimal viewpoint.

A second application of parameter optimization for volume visualization is the estimation and automatic selection of "meaningful" transfer functions (TFs). Selecting a TF is a difficult task, as the usefulness of the features that are shown or hidden by the TF highly depend on the expectations of the user. For an overview on different options on how to define TFs, we refer to the report by Ljung *et al.* [Lju+16]. Instead of fully automating the process, many works aid the user in defining TFs. For example, Kniss *et al.* [KKH01; KKH05] provide interaction widgets to aid in TF design. Haidacher *et al.* [Hai+10] extract statistics of local neighborhoods around each voxel and provide tools to design TFs in this statistical function space. For a semi-automatic process, Correa and Ma [CM10] describe visibility-driven TFs. Here, the user specifies a target visibility for certain features and an optimization routine than adapts the TF so that this target visibility is reached. To aid in this optimization, a histogram of the density values weighted by the visibility of each contributing voxel is computed to identify occlusion patterns. This process is extended by Ruiz *et al.* [Rui+11] to generate TFs in a fully automatic process. The authors present analytic approximations of the derivatives of the visibility histograms that allows a gradient-based optimization of the transfer function. Recently, Berger *et al.* [BLL19] have replaced the entire rendering pipeline by a neural network. This especially enables differentiability of the image with respect to the transfer functions, which is exploited by the authors to explore the TF space or analyze the sensitivity of the image when changing the TF.

2.5 Controlled-Precision Volume Rendering

The optical model behind direct volume rendering is the so-called *emission-absorption model* [DCH88; Lev88; Max95], see also Sec. 3.1.2. In the general case, this model can only be solved via quadrature, leading to approximation errors [WM92]. In early works, Novins and Arvo [NA92] have presented quadrature methods with a fixed error bound for the special case of pre-classification, i.e. when an emission-absorption field is given. For post-classification, Novins *et al.* [NAS92] propose a recursive refinement procedure using interval arithmetic, where segments of the ray with the highest estimated error are selected and refined. In this way, large uniform segments can be skipped, however, it requires an estimation of the minimum and maximum intensity and transparency per segment. This renders the approach unsuitable for the use of high-frequent TFs, as the intervals stored per cell become a very crude approximation once optical properties are mapped via high-frequency TFs. Similarly, Campagnolo *et al.* [CCF15] propose an iterative, adaptive Simpson quadrature scheme to evaluate the volume rendering integral up to a certain accuracy. However, to avoid arbitrarily many subdivisions when sharp peaks in the TF occur, an upper bound on the number of subdivisions needs to be considered. Etiene *et al.* [Eti+13] conduct a study on the interaction between the step size in the numerical integration and the quality of the resulting image. They propose a verification scheme by refining the screen resolution, voxel resolution or integration step size, and comparing the results from the refinement with the original images.

Under the assumption that the density function along the ray varies piecewise linearly, further controlled-precision algorithms are possible. Piecewise linear density profiles are, e.g., given on tetrahedral grids, but not on regular hexahedral (voxel) grids. On hexahedral grids, the density profile along a ray is given as a piecewise cubic function instead. The early work by Williams and Max [WM92] shows that in the case of a piecewise linear density profile with the additional assumption of a linear TF, analytic solutions are available. Kniss *et al.* [Kni+03] extend this idea to TFs defined as a sum of Gaussian functions and also extend it to multi-dimensional TFs. Pre-integrated TFs [RKE00; EKE01] pre-compute a 2D matrix with the solutions of the emission-absorption model if starting at density d_1 and ending at density d_2 with a fixed step size. To compute this matrix, quadrature is needed again. But since this computation is independent of the rendering process, a slow, but high-quality quadrature scheme can be used. Pre-integration is exact under the assumption of piecewise linear densities as well. On hexahedral grids, this method again introduces approximation errors. We show in Paper C [WW21], see Sec. 3.1.3, how controlled-precision rendering under weaker assumptions – trilinear interpolation on hexahedral grids and a piecewise polynomial function – can be achieved.

In a different direction, Lindholm *et al.* [Lin+13] split the view ray not in constant steps in world space, but rather at control points of the (piecewise linear) TF. This is performed by rendering semi-transparent isosurfaces at densities given by the TF control points using rasterization. To control the

accuracy of this method, additional isosurfaces are introduced between the control points of the TF, i.e. the TF is subdivided. With this method, sharp spikes in the TF are captured, but for smooth TFs, a large number of isosurfaces are needed for convergence. Furthermore, additional approximation errors are introduced when rendering the isosurface via Marching Cubes [LC87; TPG99]. Marching cubes discretizes the piecewise cubic isosurface into piecewise linear, i.e. flat, triangles.

3

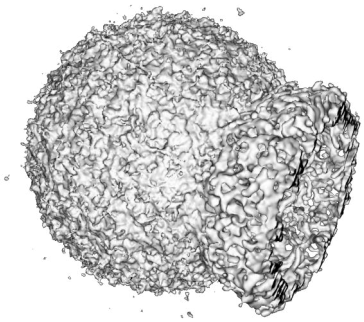
Fundamentals and Methods

The publications contributing to this thesis build upon several prior concepts. In this chapter, we introduce the theoretical background of the methods used, applications, and extensions thereof, as well as the core ideas of the methods proposed in this thesis. We start with the techniques for volume visualization in Sec. 3.1 concluded with the core idea on how to achieve volume rendering with a controlled precision in Sec. 3.1.3. The concepts of automatic differentiation and the application to differentiable volume rendering are introduced in Sec. 3.2. Neural networks, both fully-connected networks and convolutional networks, are summarized in Sec. 3.3, leading to the proposed super-resolution methods for accelerating volume rendering.

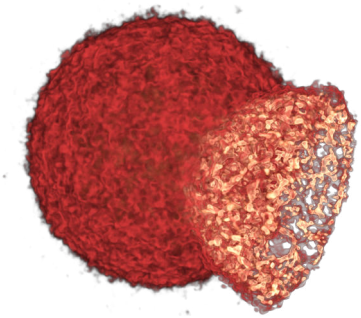
3.1 Volume Visualization

The task of volume visualization is to convert a three-dimensional volume into one or more two-dimensional images that are easier for the human user to interpret. Typically, this first involves a data filtering stage, where a user interactively specifies the features to display. This is then followed by a projection stage where the filtered volume is rendered onto a 2D image.

Let $V : \Omega \to \mathbb{R}$ be the scalar volume that should be visualized. This function maps from positions in $\Omega \subset \mathbb{R}^3$ to density values that are typically normalized in the range $[0, 1]$. In our publications we assume V to be given on a regular hexahedral grid with the data stored at the *vertices* $V_{i,j,k}$. Eight vertices define the corners of a *cell* or voxel, and trilinear interpolation $V_{ijk}(\alpha, \beta, \gamma)$ is used within, defined in (3.1) and visualized in Fig. 3.2.



(a) Isosurface Rendering: A single surface is rendered and shaded using a Lambertian model.



(b) Direct volume rendering: Each density value is mapped to a color and absorption and accumulated for each pixel.

Figure 3.1: Rendering of the same dataset, the Ejecta simulation, using (a) isosurface rendering and (b) using direct volume rendering.

$$
\begin{aligned}
V_{ijk}(\alpha,\beta,\gamma) = \; & (1-\alpha)(1-\beta)(1-\gamma)\; V_{ijk} + \\
& (1-\alpha)(1-\beta)\gamma\; V_{i,j,k+1} + \\
& (1-\alpha)\beta(1-\gamma)\; V_{i,j+1,k} + \\
& \vdots \\
& \alpha\beta\gamma\; V_{i+1,j+1,k+1}
\end{aligned}
\tag{3.1}
$$

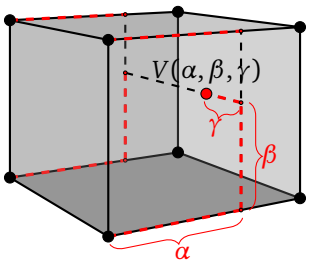


Figure 3.2: Trilinear interpolation within a voxel.

The values $\alpha, \beta, \gamma \in [0,1]$ specify the fractional position within the current cell. Such voxel representations were described as early as 1983 for the purpose of medical imaging [UH83; Rey+85]. Besides trilinear interpolation, higher-order filters are also commonly used. We refer to Marschner and Lobb [ML94] for an evaluation of interpolation methods. The two main approaches to visualize such density grids are *isosurface rendering* (Sec. 3.1.1, Fig. 3.1a) and *direct volume rendering* (Sec. 3.1.2, Fig. 3.1b). The former displays a single feature or surface, the latter displays a volumetric impression.

3.1.1 Isosurface Rendering

Let ρ be a user-specified density value. This value defines the surface

$$S_\rho := \{\boldsymbol{x} \in \Omega : V(\boldsymbol{x}) = \rho\} \tag{3.2}$$

that should be visualized in Isosurface Rendering. There exist two classes of algorithms to display this surface S_ρ, rasterization-based and raytracing-based algorithms.

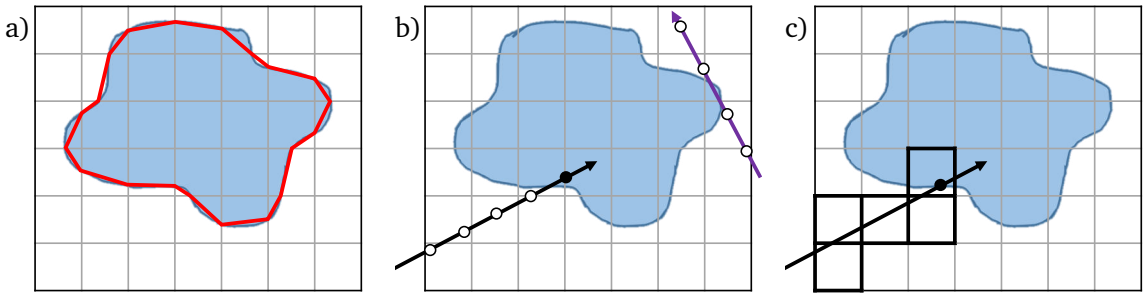


Figure 3.3: An isosurface stored on a grid, shown here as a 2D slice in blue, is rendered using three different algorithms. a) Marching cubes, b) Raytracing with a constant step size, c) voxel traversal with analytic intersection tests.

The first rasterization method is *Marching Cubes* [LC87], see Fig. 3.3a. Here, the surface is discretized into triangles and then rendered using the traditional rasterization pipeline on the graphics card. The triangle discretization follows the following procedure:

1. Classify each of the eight vertices of the current cell as either inside ($V_{ijk} \geq \rho$) or outside ($V_{ijk} < \rho$) and build an eight-bit index from that.
2. Using a table lookup, determine the list of edges, out of the 12 possible edges, that hold an intersection.
3. For each such edge, compute the position of the surface-edge intersection.
4. Assemble the triangles into a triangle soup and render them using the standard rasterization pipeline.

This whole process discretizes the – in the case of trilinear interpolation – cubic surface per cell into piecewise linear surfaces. This leads to an over- and underestimation of the surface as can be seen in Fig. 3.3a. The Marching Cubes algorithm suffers also from the so-called ambiguity problem. For certain cell configurations, there are multiple ways how the vertices at the cell edges are connected into triangles and an incorrect choice leads to gaps between neighboring cells. One way to resolve this issue is to use the *Asymptotic Decider* [NH91] which requires extra conditionals, but leads to watertight meshes. Alternative, *Marching Tetrahedra* [TPG99] resolves the issue by subdividing the cell into several tetrahedras in which the surface can be discretized into triangles without ambiguities.

For raytracing-based methods, a ray

$$s(t) = \boldsymbol{x}_0 + t\,\boldsymbol{\omega} \tag{3.3}$$

with start position $\boldsymbol{x}_0$ and direction $\boldsymbol{\omega}$ is shot from the camera through a pixel of the screen into the scene [App68]. Then, the intersection of the ray with the isosurface S_ρ is computed and returned. The simplest method (Fig. 3.3b) is to walk along the ray with a fixed Δt step by step, evaluate the density

at step i, $V(\boldsymbol{x}_0 + i\Delta t\boldsymbol{\omega})$, and terminate once the interpolated density exceeds the threshold ρ [Lev88; Lev90a; Tie+90; Wei+21]. This method is simple to implement and fast to render due to the hardware support for trilinear interpolation but can produce incorrect results. In Fig. 3.3b, the intersection at the black ray is detected too late as the ray has already entered the object when the next sample is evaluated. Furthermore, intersections can be missed if the ray skims a corner of the object (purple ray in Fig. 3.3b). The former error can be lessened by using binary search to refine the intersection.

To guarantee, that no intersections are missed, the trilinear interpolation scheme within a cell has to be utilized explicitly, see Fig. 3.3c. First, instead of performing fixed stepping, we walk through the volume using a voxel traversal algorithm [AW87; JTC14]. This leads to an ordered sequence of visited voxels with their entry and exit time t. Then, inserting the ray equation (3.3) into the trilinear interpolation (3.1) per visited voxel gives rise to a cubic equation in t,

$$V_{ijk}(s(t)) = v_0 + v_1 t + v_2 t^2 + v_3 t^3 = \rho, \tag{3.4}$$

that can be solved analytically [LC96; Par+98; Neu+02; Mar+04]. Due to the analytical computation of the ray-isosurface intersection, no intersection can be missed.

3.1.2 Direct Volume Rendering

For direct volume rendering (DVR), we follow the low-albedo emission-absorption model as presented by Max [Max95]. Let $\tau : [0,1] \to \mathbb{R}_0^+$ be the absorption due to a given density, and $C : [0,1] \to \mathbb{R}_0^+$ the assigned color, both specified via a transfer function. By convention, the self-emission is then given by $g(d) = C(d)\tau(d)$ with the the density $d \in [0,1]$ obtained by interpolating the volume V. This coupling of the emission to the absorption has the intuition that in regions with more material, i.e. more absorption, the emission scales accordingly. The transfer function is a user-defined mapping that allows to interactively select and highlight different features of a dataset. An example of such a mapping is shown in Fig. 3.4.

The transparency of the line segment from $t = a$ to b is defined as

$$T(a,b) = \exp\left(-\int_a^b \tau(V(s(t)))\mathrm{d}t\right). \tag{3.5}$$

The transparency is 1 if the medium between a and b does not absorb any light and approaches zero for complete absorption. Then, the light intensity reaching the eye is

$$L(a,b) = \int_a^b g(V(s(t)))T(a,t)\mathrm{d}t, \tag{3.6}$$

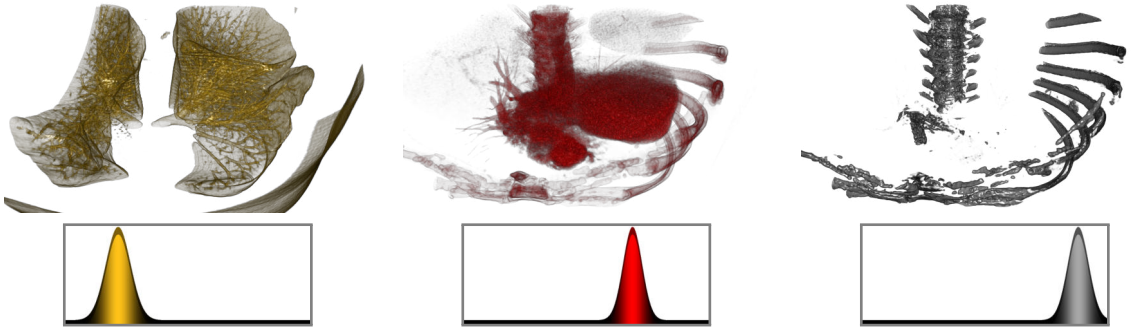


Figure 3.4: A user-defined transfer function (bottom row) maps from the density on the x-axis to color and absorption on the y-axis. This way, different features of the Thorax dataset can be highlighted.

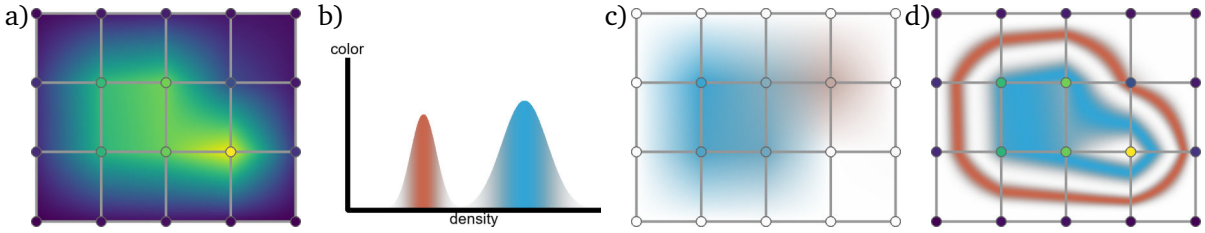


Figure 3.5: Demonstration of pre-classification vs. post-classification. a) A volume is given by the densities on the vertices together with an interpolation function. b) This TF should now be applied. c) In pre-classification, the TF is applied to the densities at the vertices and the resulting colors are interpolated. This leads to a smooth color distribution with the frequency of the features limited by the grid resolution. d) In post-classification, the TF mapping is applied after the density interpolation, leading to much finer details.

were $g(d) = C(d)\tau(d)$. Usually, the emission is not given as a single scalar intensity, but as an RGB tuple. In this case, (3.6) is evaluated component-wise for the red, green, and blue channels.

Note that in the above equations, the volume is interpolated first and the densities are mapped to absorption and color via the transfer function *after* the interpolation. This is called *post-classification* [EKE01] and allows the introduction of features with a finer resolution than the grid using the TF. The alternative is *pre-classification*, where the TF is applied *before* the interpolation directly on the vertices of the grid. A demonstration of of the differences between pre-classification and post-classification is shown in Fig. 3.5. Albeit post-classification is the "correct" way to apply the TF [EKE01], pre-classification has found applications, e.g., in quadrature schemes with a controlled error bound, see Sec. 3.1.3.

The general volume integrals, (3.5) and (3.6), do not have a simple closed form for arbitrary functions τ and g. Therefore, quadrature methods are employed, where the domain $[a, b]$ is subdivided into smaller parts and integrated separately [Max95]. For N subdivisions of $[a, b]$, the boundaries are denoted by $a = t_0 < t_1 < ... < t_N = b$.

Transparency First, the transparency integral for $T(a,b)$, see (3.5), can be split into smaller parts as following:

$$\begin{aligned} T(a,b) &= \exp\left(-\int_a^b \tau(V(s(t)))\mathrm{d}t\right) \\ &= \exp\left(-\sum_{i=0}^{N-1}\int_{t_i}^{t_{i+1}} \tau(V(s(t)))\mathrm{d}t\right) \\ &= \prod_{i=0}^{N-1} \underbrace{\exp\left(-\int_{t_i}^{t_{i+1}} \tau(V(s(t)))\mathrm{d}t\right)}_{=T(t_i,t_{i+1})=:T_i}. \end{aligned} \tag{3.7}$$

Hence, the final transparency is given by $T = \prod_{i=0}^{N-1} T_i$ where 1 indicates fully transparent, 0 fully opaque. Alternatively, one is usually more interested in the opacity α with 1 being fully opaque and 0 being fully transparent. The relationship between α and T follows

$$\alpha := 1 - T \Leftrightarrow T = 1 - \alpha. \tag{3.8}$$

Substituting (3.8) into (3.7) gives rise to

$$\begin{aligned} 1-\alpha &= \prod_{i=0}^{N-1}(1-\alpha_i) \text{ with } \alpha_i = 1 - T_i \\ \Leftrightarrow \alpha &= 1 - \prod_{i=0}^{N-1}(1-\alpha_i). \end{aligned} \tag{3.9}$$

Let $\alpha^{(k)} := 1 - \prod_{i=0}^{k-1}(1-\alpha_i)$ be the evaluation of (3.9) up to k terms. By induction we have:

$$\begin{aligned} k = 0 &: \alpha^{(0)} = 0 \\ k \to k+1 &: \alpha^{(k+1)} = \alpha^{(k)} + (1-\alpha^{(k)})\alpha_k. \end{aligned} \tag{3.10}$$

This leads to the well-known front-to-back algorithm:

Algorithm 1 Front-to-back algorithm for absorption

1: $\alpha = 0$
2: **for** $i = 0, ..., N-1$ **do**
3: evaluate $\alpha_i = 1 - T_i$
4: $\alpha = \alpha + (1-\alpha)\alpha_i$
5: optional early-out if α gets close to 1
6: **end for**

The important aspect in this algorithm is to compute α_i. In the simplest form, the following approximation schemes, summarized by Max [Max95], are used:

$$\begin{aligned}
\alpha_i &= 1-\exp\left(-\int_{t_i}^{t_{i+1}} \tau(V(s(t)))\mathrm{d}t\right) \\
&\approx 1-\exp(-\tau(V(s(t_i)))\underbrace{(t_{i+1}-t_i)}_{=:\Delta t_i}) \quad \text{by approximating the left factor} \\
&\approx 1-\max(0,\, 1-\Delta t_i\,\tau(V(s(t_i)))) \qquad \text{by Taylor expansion} \\
&= \min(1,\, \Delta t_i\,\tau(V(s(t_i)))) \qquad .
\end{aligned} \tag{3.11}$$

In special cases, e.g. hexahedral grids with tri-linear interpolation, analytical solutions for the transparency are possible, as exploited in our work, see Paper C.

Emission Next, we analyze the emission term L, based on the emission coefficients $g(d) = C(d)\tau(d)$, again subdivided over intervals $t_0, ..., t_N$.

$$\begin{aligned}
L(a,b) &= \int_a^b g(V(s(t)))T(a,t)\mathrm{d}t \\
&= \sum_{i=0}^{N-1}\int_{t_i}^{t_{i+1}} g(V(s(t)))T(a,t)\mathrm{d}t \\
&= \sum_{i=0}^{N-1} T(a,t_i)\underbrace{\int_{t_i}^{t_{i+1}} g(V(s(t)))T(t_i,t)\mathrm{d}t}_{=:L_i}.
\end{aligned} \tag{3.12}$$

Hence the final emission is given by $L = \sum_{i=0}^{N-1} T(a,t_i)L_i$ where $T(a,t_i) = \prod_{j=0}^{i-1} T_j$, see (3.7). Using (3.9), we arrive at

$$L = \sum_{i=0}^{N-1}(1-\alpha^{(i)})L_i \tag{3.13}$$

which gives rise to the following extension of Algorithm 1, now incorporating emission, shown in Algorithm 2.

Note that the emission here is presented as a scalar quantity, but the computations can be easily extended to vector quantities, e.g. RGB-colors or spectra. Again, the crucial part of this algorithm is

Algorithm 2 Front-to-back algorithm for absorption and emission

1: $\alpha = 0, L = 0$
2: **for** $i = 0, ..., N-1$ **do**
3: evaluate $\alpha_i = 1 - T_i$ and L_i
4: $L = L + (1-\alpha)L_i$
5: $\alpha = \alpha + (1-\alpha)\alpha_i$
6: optional early-out if α gets close to 1
7: **end for**

the computation of L_i, we will again present the simple approximation, commonly used in rendering here [Max95]:

$$\begin{aligned} L_i &= \int_{t_i}^{t_{i+1}} g(V(s(t)))T(t_i,t)\mathrm{d}t \\ &\approx (g(V(s(t_i)))\underbrace{T(t_i,t_i)}_{=0})(\underbrace{t_{i+1}-t_i}_{=:\Delta t_i}) \quad \text{by approximating the left factor} \\ &= g(V(s(t_i)))\Delta t_i \end{aligned} \tag{3.14}$$

3.1.3 Controlled-Precision Volume Rendering

As stated above, for the general absorption-emission integral (3.12), no analytical solutions are known. This is mostly due to the trilinear interpolation, which gives rise to a piecewise cubic function of the density along the ray. Albeit analytical solutions exist for special cases.

If instead of a hexahedral grid made out of cubical voxels, a tetrahedral grid is used, the volume interpolation within a tetrahedral cell leads to a linear function of the density [EKE01; RKE00]. For this case, methods that assume a piecewise linear function of the density, are exact. This includes pre-integrated transfer functions by Engel *et al.* [EKE01] and Gaussian transfer functions by Kniss *et al.* [Kni+03].

For trilinear interpolation, quadrature schemes are needed. Besides the simple approximation in (3.14), Novins and Arvo [NA92] and de Boer *et al.* [Boe+97] introduce higher-order quadrature schemes that allow to control the maximal allowed error. These works, however, assume a pre-classified volume. That is, the transfer function is applied before the interpolation, giving rise to an absorption-emission volume that is then interpolated. Recall in Sec. 3.1.2, volume rendering was introduced using post-classification. The transfer function $\tau(d), L_e(d)$ is applied after the trilinear interpolation $V(\boldsymbol{x})$. This allows for more high-frequent transfer functions, as the frequency of the color variations due to the TF are not limited to the spatial resolution of the grid.

We propose (Paper C, [WW21]) how to extend the ideas by Novins and Arvo [NA92] to the post-classification case for trilinearly interpolated grids, i.e., the density along the ray per voxel cell $V(s(t))$ is a cubic function. We assume, that the transfer function $\tau(d), C(d)$ is given as a piecewise polynomial

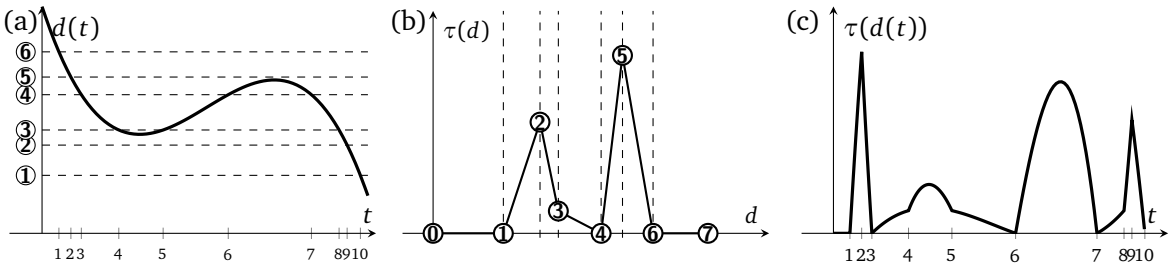


Figure 3.6: (a) Tracing a ray $s(t)$ through a tri-linearly interpolated voxel $V(\boldsymbol{x})$ gives rise to cubic polynomial of the density $V(s(t))$. (b) We assume, that the transfer function $\tau(d), L_e(d)$ is given as a piecewise polynomial function in the density. Here we use a piecewise linear function. (c) Then, applying the transfer function to the density field in a post-classification setting gives rise to a piecewise cubic function of the absorption and emission $\tau(V(s(t))), L_e(V(s(t)))$.

function of degree k. Then the central observation is, that the application of the transfer function to the trilinear interpolation $\tau(V(s(t))), C(V(s(t)))$ is a piecewise polynomial function of degree $k+2$. This case is visualized in Fig. 3.6: applying a piecewise linear TF (Fig. 3.6b) to the cubic function of the density (Fig. 3.6a) gives rise to a piecewise cubic function (Fig. 3.6c).

To construct the resulting piecewise polynomial function, we can look at the problem in the following way: The control points d_i of the TF defining the polynomial segments, numbered 0 to 7 in Fig. 3.6b, represent isovalues in the volume. This leads to the following algorithm for controlled-precision volume rendering, slightly simplified:

1. Walk through the volume voxel-by-voxel using a voxel traversal algorithm [AW87; Hoe16]. This gives an ordered sequence of visited voxels per ray and the respective entry and exit times $t_{\text{in}}, t_{\text{out}}$.

2. Per voxel, construct the cubic polynomial of the density values, see (3.4), using the algorithm by Parker *et al.* [Par+98]
$$V(s(t)) = v_0 + v_1 t + v_2 t^2 + v_3 t^3. \tag{3.15}$$

3. Solve for all intersections with the isosurfaces of the TF control points d_i, up to three isosurface intersections per control point can exist. There exist closed-form solutions for the roots of a cubic equation. Examples include Cardano's formula [Sch13; LC96] and Viète's formula [Nic06]. We found, however, that the iterative algorithm by Marmitt *et al.* [Mar+04] produces faster and more accurate intersections. The result is a sequence of segments along the ray, see Fig. 3.6a with $N = 10$,
$$t_{\text{in}} < t_1 < \ldots < t_N < t_{\text{out}}. \tag{3.16}$$

4. Per segment, the TF is polynomial, or in the example of Fig. 3.6b linear, i.e. $\tau(d) = \tau_0 + \tau_1 d$. Then we can express the absorption and color of a ray segment t_i, t_{i+1} as a cubic polynomial, Fig. 3.6c,

$$\begin{aligned} \tau(V(s(t))) &= \bar{\tau}_0 + \bar{\tau}_1 t + \bar{\tau}_2 t^2 + \bar{\tau}_3 t^3 \\ C(V(s(t))) &= \bar{C}_0 + \bar{C}_1 t + \bar{C}_2 t^2 + \bar{C}_3 t^3 \end{aligned}. \tag{3.17}$$

5. Now we follow the method Novins and Arvo [NA92], applied per such segment above instead of per voxel. The integral of the transparency $T(t_i, t_{i+1})$ can be solved analytically

$$\begin{aligned} T(t_i, t_{i+1}) &= \exp\left(-\int_{t_i}^{t_{i+1}} \tau(V(s(t)))\mathrm{d}t \right) \\ &= \exp\left(-\int_{t_i}^{t_{i+1}} \bar{\tau}_0 + \bar{\tau}_1 t + \bar{\tau}_2 t^2 + \bar{\tau}_3 t^3 \mathrm{d}t \right) \\ &= \exp\left(\underbrace{-\left[\mathcal{T}_0 + \mathcal{T}_1 t + \mathcal{T}_2 t^2 + \mathcal{T}_3 t^3 + \mathcal{T}_4 t^4 \right]_{t_i}^{t_{i+1}}}_{=:\mathcal{T}_{t_i}(t_{i+1})} \right) \end{aligned}. \tag{3.18}$$

   For the emission on the given segment, $g(t) = C(V(s(t)))\tau(V(s(t)))$ is a polynomial of degree six, obtained by polynomial multiplication of the expressions in (3.17). Then the integral of the emission takes on the following form,

$$\begin{aligned} L(t_i, t_{i+1}) &= \int_{t_i}^{t_{i+1}} C(V(s(t)))\tau(V(s(t)))T(t_i, t_{i+1})\mathrm{d}t \\ &= \int_{t_i}^{t_{i+1}} g(t)\exp(\mathcal{T}_{t_i}(t))\mathrm{d}t \end{aligned}. \tag{3.19}$$

   Integrals of the form polynomial-exponential-polynomial have no analytical solutions. Hence, we apply the quadrature schemes (Trapezoid rule and Simpson's rule) presented by Novins and Arvo [NA92] to evaluate the integral up to a user-defined error.

The proposed method allows to perform direct volume rendering to a higher precision. This is especially noticeable for datasets with a sharp transfer function. An example can be seen in Fig. 3.7 for the "Tube" dataset [WW21], a volume given by the function

$$v(x, y, z) = 10(0.1 - \sqrt{y^2 + z^2}(0.9 - 0.5\cos(7x))^3). \tag{3.20}$$

In Fig. 3.7a, the dataset was rendered with ray marching using a constant step size as introduced in Sec. 3.1.2. Quadrature errors due to a too coarse step size are noticeable in the form of "rings". The

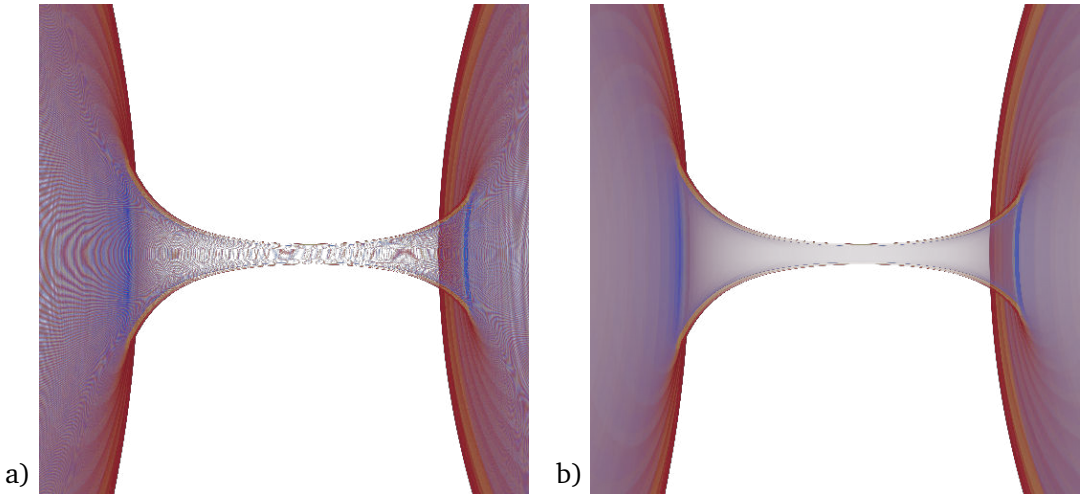


Figure 3.7: Renderings of the "Tube" dataset with a TF exhibiting narrow peaks. (a) Ray marching with a constant step size as presented in Sec. 3.1.2 gives rise to quadrature errors in the form of ringing artifacts. (b) The controlled-precision rendering method presented in Sec. 3.1.3 can resolve the image without noticeable quadrature errors.

controlled-precision rendering method as presented above can successfully resolve the high-frequent peaks of the TF and can render the image without noticeable errors (Fig. 3.7b). In Paper C [WW21], we further show how to extend this approach to scale-invariant direct volume rendering as proposed by Kraus [Kra05].

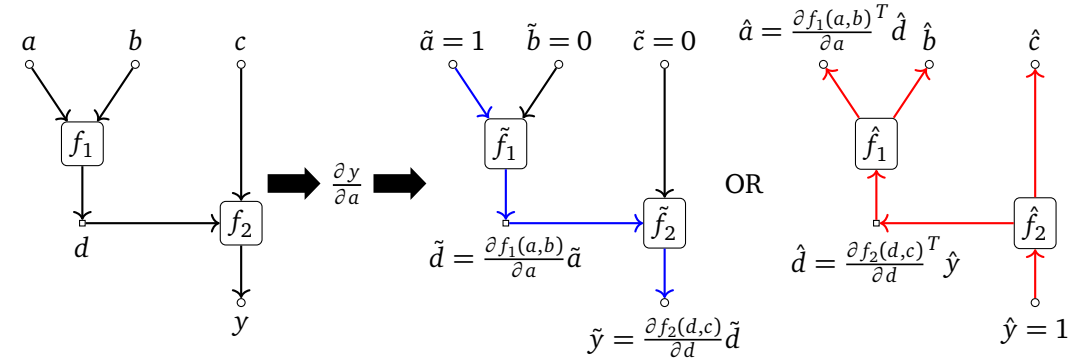
3.2 Automatic Differentiation

For training neural networks (Sec. 3.3, Paper A and Paper B) or solving inverse problems, algorithms have to be made differentiable. An example of an inverse problem is the reconstruction of the volume density V from images through the rendering process (Sec. 3.1.2), part of Paper D. While it is possible to manually determine the expression of the derivative, often it is faster and less error-prone to automatically evaluate the derivatives. This is the task of Automatic Differentiation (AD). For an extended mathematical introduction of AD, we refer to the book by Griewank and Walther [GW08]. Here we follow the more practical-oriented description by Bartholomew-Biggs *et al.* [BB+00].

Assume that a program is represented as a *Wengert list* [Wen64], which can be seen as the sequential trace of primitive operations. A Wengert list with P inputs, O outputs and N operations has the form given in Algorithm 3. All states, inputs, outputs and intermediate results, are stored in the state variables x_i. The N primitive operations can take all P inputs and all previously computed states as argument and produce a new scalar state. The last O states are returned as output. For simplicity, the functions operate on scalar values, but the description below can be easily extended to vector

Algorithm 3 Wengert list representation of an arbitrary program with P inputs, O outputs and N operations.

Input $x_1, x_2, ..., x_P$
for i from 1 to N **do**
  $x_{i+P} := f_i(x_1, ..., x_{P+i-1})$
end for
Output $y_i = x_{P+N-O+i}, i = 1, ..., O$



a) Regular Evaluation b) Forward Differentiation c) Adjoint Differentiation

Figure 3.8: Visualization of the evaluation of the function $y = f_2(f_1(a,b),c)$ for demonstrating automatic differentiation. a) The regular evaluation, (b) forward differentiation, and (c) adjoint differentiation for $\frac{\partial y}{\partial a}$.

quantities. Now the goal of automatic differentiation is to evaluate the derivatives $\partial y_j / \partial x_i$ using the chain rule. An example of a simple program consisting of two operations and three inputs is shown in Fig. 3.8.

3.2.1 Forward Differentiation

In *Forward Differentiation* [Nei10; BB+00], the derivatives are propagated alongside the regular program evaluation (Fig. 3.8b). Without loss of generality, assume that derivatives with respect to only a single input x_k shall be computed. Let the *forward variables* be defined as

$$\begin{aligned} &\tilde{x}_i = \frac{\partial x_i}{\partial x_k}, \text{ initialized at the inputs as:} \\ &\tilde{x}_i = \frac{\partial x_i}{\partial x_k} := \begin{cases} 1, i = k \\ 0, \text{otherwise} \end{cases}, i \in \{1, ..., P\}. \end{aligned} \tag{3.21}$$

In other words, the forward variables carry the derivative of the current state with respect to the input to derive for. Then, the derivatives are propagated through the Wengert list by replacing each f_i by the respective forward function $\tilde{f}_i$:

$$\tilde{x}_i = \tilde{f}_i(\tilde{x}_1, ..., \tilde{x}_{i-1}) := \frac{\partial f_i}{\partial x_1}\tilde{x}_1 + ... + \frac{\partial f_i}{\partial x_{i-1}}\tilde{x}_{i-1}. \tag{3.22}$$

The forward variables for the output then contain the final derivatives. Using Forward Differentiation, the derivatives are propagated jointly with the regular program evaluation. This can be realized, e.g., using operator overloading as demonstrated in Paper D [WW22] and shown below. The advantage of Forward Differentiation is its relative simplicity of implementation. The disadvantage, however, is, that the computations scale linearly with the number of inputs to differentiate, as the procedure above has to be repeated for every input. Therefore, this approach is best used in situations with large number of operations N, many outputs O, but only a few inputs P.

We now summarize, how Forward Differentiation can be implemented using operator overloading in simplified C++. We demonstrate this for the example of front-to-back blending with a grayscale color instead of a full RGB-color for simplicity. Here, the accumulated opacity `acc_opacity` and color `acc_color` up to the current sample are blended with the current contribution `curr_opacity`, `curr_color` and give rise to the new opacity and color `new_opacity, new_color`.

```
blend(float acc_opacity, float acc_color, float curr_opacity, float curr_color) {
    float new_color = acc_color + (1-acc_opacity)*curr_color;
    float new_opacity = acc_opacity + (1-acc_opacity)*curr_opacity;
    return new_opacity, new_color;
}
```

To propagate gradients through the function, we first define the custom datatype that will hold the derivatives.

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

Here, the template parameter `T` holds the underlying datatype, e.g. `float`. The number of derivatives that are propagated at the same time is specified via `p`. For constant values, the derivatives are initialized with zeros. For a parameter that shall be differentiated, the entry of `derivatives` at the corresponding index is set to one, see also (3.21).

To realize the forward functions (3.22) we provide operator overloadings for all common arithmetic operations. For example, addition and multiplication are implemented as following:

```
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.derivative[i] + b.derivative[i];
    }
    return c;
}
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;
}
```

To support operations like `(1-acc_opacity)`, where one of the arguments is a single scalar and not an instance of `fvar`, we additionally provide operator overloadings with the first or second argument being a regular scalar value:

```
template<typename T, int p>
fvar<T, p> operator+(T a, fvar<T, p> b) {...}
template<typename T, int p>
fvar<T, p> operator+(fvar<T, p> a, T b) {...}
```

Finally, we modify each function in the rendering code to accept any datatype as input

```
template<typename T1, typename T2, typename T3, typename T4>
auto blend(T1 acc_opacity, T2 acc_color, T3 curr_opacity, T4 curr_color) {
    auto new_color = acc_color + (1-acc_opacity)*curr_color;
    auto new_opacity = acc_opacity + (1-acc_opacity)*curr_opacity;
```

```
    return new_opacity, new_color;
}
```

This allows to use the same function for a regular program evaluation without gradients (`T1,T2,T3,T4` are of type `float`), or with gradient propagation (one of the `T`'s is of type `fvar`). The compiler can automatically deduce the correct datatypes of the intermediate variables and return value with the special type `auto` and uses the correct operator overloading.

3.2.2 Adjoint Differentiation

Using Adjoint Differentiation, the chain rule is evaluated in reverse order, see Fig. 3.8c for an example. Due to concurrent developments, this method is known under many different names: Adjoint Differentiation, the Adjoint Method or Reverse Accumulation in control theory [Dre62; McN+04], or Backpropagation in the context of neural networks [RHW85; RHW86].

Let us assume that the program in Algorithm 3 only produces a single scalar output $y := x_{P+N+O}$ ($O = 1$). This is the usual case for optimization if a multi-dimensional function should be maximized or minimized with respect to a specific cost function or distance metric. Let the *adjoint variables* be defined as

$$\hat{x}_i = \frac{\partial y}{\partial x_i}, \text{ initialized as:}$$
$$\hat{x}_i = \begin{cases} 1, i = P+N+O \text{ (the output)} \\ 0, \text{otherwise} \end{cases}. \tag{3.23}$$

In other words, the adjoint variables carry the derivative of the output with respect to the current state variable. Therefore, the target derivatives $\partial y / \partial x_i$ are stored in the adjoint variables of the inputs. This leads to a reverse evaluation, where sequence of functions in Algorithm 3 is executed from $i = N$ to $i = 1$. Each function f_i is replaced by the respective adjoint function that accumulates the derivatives for the inputs to that function,

$$\hat{x}_j \mathrel{+}= \left(\frac{\partial f_i}{\partial x_j} \right)^T \hat{x}_i, j = 1, ..., i-1. \tag{3.24}$$

The derivatives stored in the adjoint variable of the output of f_i are thus propagated to the adjoint variables of the inputs. Transposing the partial derivative is important if dealing with vector-valued arguments instead of scalar values. The Adjoint method leads to two implications: First, for non-linear functions f_i, the partial derivative is not constant. Therefore, the input states for each function f_i have to be stored for the adjoint evaluation in (3.24). This leads to memory requirements that scale with the number of operations, a typical concern when training deep neural networks. Second, derivatives with respect to the inputs are only computed if they are actually needed in the current function f_i, i.e. the

Algorithm 4 Front-to-back algorithm for absorption, revisited

$\alpha^{(0)} = 0$
for $i = 0, 1, ..., N-1$ **do**
    evaluate α_i
    store $\alpha_i, \alpha^{(i)}$
    $\alpha^{(i+1)} = \alpha^{(i)} + (1-\alpha^{(i)})\alpha_i$
end for
store $\alpha^{(N)}, \alpha_{\text{GT}}$
$e = (\alpha^{(N)} - \alpha_{\text{GT}})^2$

Algorithm 5 Adjoint code for the front-to-back algorithm to compute the $\hat{\alpha}_i$'s

$\hat{e} = 1$ initialize adjoint variable
fetch $\alpha^{(N)}$, α_{GT} from storage
$\hat{\alpha_N} = 2(\alpha^{(N)} - \alpha_{\text{GT}})\hat{e}$
for $i = N-1, ..., 1, 0$ **do**
    fetch $\alpha_i, \alpha^{(i)}$ from storage
    $\hat{\alpha}_i = (1-\alpha^{(i)})\hat{\alpha}^{(i+1)}$
    $\hat{\alpha}^{(i)} = (1-\alpha_i)\hat{\alpha}^{(i+1)}$
end for

respective partial derivative is non-zero. This is beneficial as in typical scenarios, the input parameters are only needed at a few operations in the algorithm. For example, network weights of neural networks are only required in the evaluation of their respective layer. For volume reconstructions, the volume density of a specific point (the input to optimize) is only needed when a ray hits that location.

Due to its scalability to many input parameters, the Adjoint method is the most common choice for automatic differentiation if more than a few parameters should be optimized at once. It is implemented in popular deep learning frameworks like PyTorch [Pas+19], TensorFlow [Aba+16], or in differentiable renderers like Mitsuba 2 [ND+19]. Operator overloading to propagate the gradients as used in Forward Differentiation, see Sec. 3.2.1 is not easily possible with the Adjoint method due to the inversion of the computation order. The deep learning frameworks and Mitsuba 2 instead build a computation graph, see Fig. 3.8. If a function is computed, a descriptor of that function together with the inputs is stored as a node in the graph. After the main program has been evaluated and gradients should now be computed, this graph is queried and the operations are repeated in reverse order. During this reverse evaluation, the adjoint function for each node in the graph is executed.

As a concrete example, we demonstrate how the Adjoint method would be applied to front-to-back blending for the absorption, see Algorithm 1 in Sec. 3.1.2. Assume that we have some target absorption α_{GT} that should be achieved. Differences in the current absorption α to the target are penalized by the squared error. To optimize the sampled absorptions α_i along the ray, see (3.11), we want to compute derivatives $\hat{\alpha}_i$ of those samples. Concrete optimization strategies are later introduced in the section on neural networks (Sec. 3.3). The revisited algorithm for front-to-back blending is shown in Algorithm 4. The resulting Adjoint code if the derivatives are computed manually is shown in Algorithm 5. Note that we introduced here the superscript $\alpha^{(i)}$ to keep track of the modified state of the accumulated absorption. In Algorithm 1, the accumulated absorption was simply denoted α and overwritten in every loop iteration.

In the computation graph, when line 5 in Algorithm 4 is executed, the inputs $\alpha^{(i)}, \alpha_i$ are stored, as well as a description of how the Adjoint code looks like. During the reverse evaluation, these inputs have to be loaded from storage (line 5) in Algorithm 5 and then the gradients are propagated (line 6-7 in Algorithm 5). This means, without further optimizations, for every operation, all inputs have to be stored. This severely limits, how large the computation graph can become, i.e. how many steps along the ray can be used.

To overcome the memory limitation mentioned above, checkpointing strategies [BB+00; GW00; Che+16] were developed where parts of the computation sequence are repeated. For the special case of differentiating the direct volume rendering integral (see Sec. 3.1.2), we show in Paper D [WW22], summarized below, how an analytic inversion of specific operations allow reducing the memory requirements to a constant, i.e. independent of the number of steps along the ray. A similar method was concurrently developed by Vicini *et al.* [VSJ21] for path tracing with multiple scattering events.

For the case of direct volume rendering, the critical operation is the front-to-back blending step. Now we use the full version including color, instead of the reduced version including only absorption as shown in Algorithm 4 and Algorithm 5. In front-to-back blending, the current accumulated opacity and color $\alpha^{(i)}, C^{(i)}$ are combined with the opacity and color α, C of the current sample to obtained the newly blended opacity and color $\alpha^{(i+1)}, C^{(i+1)}$.

$$\begin{aligned} C^{(i+1)} &= C^{(i)} + (1-\alpha^{(i)})C \\ \alpha^{(i+1)} &= \alpha^{(i)} + (1-\alpha^{(i)})\alpha. \end{aligned} \tag{3.25}$$

The adjoint code is given as

$$\begin{aligned} \hat{\alpha} = (1-\alpha^{(i)})\hat{\alpha}^{(i+1)},\ \hat{C} &= (C-\alpha^{(i)})\hat{C}^{(i+1)}, \\ \hat{\alpha}^{(i)} &= (1-\alpha)\hat{\alpha}^{(i+1)} - C\cdot\hat{C}^{(i+1)}, \\ \hat{C}^{(i)} &= \hat{C}^{(i+1)}. \end{aligned} \tag{3.26}$$

If no optimizations would be applied, all inputs $\alpha^{(i)}, C^{(i)}, \alpha, C$ would need to be stored. We show, that this storage can be avoided. During the evaluation of the Adjoint code, the current sample α, C is recomputed. This is an efficient operation as it only involves sampling the volume density and applying the transfer function. Then, the blending step (3.25) can be inverted,

$$\begin{aligned} \alpha^{(i)} &= \frac{\alpha-\alpha^{(i+1)}}{\alpha-1} \\ C^{(i)} &= C^{(i+1)} - (1-\alpha^{(i)})C. \end{aligned} \tag{3.27}$$

This allows for the reconstruction of the current accumulated opacity and color at every sample. Only the last color and opacity ($\alpha^{(N)}$ in Algorithm 4) still needs to be stored. All other accumulated colors can be recovered by repeatedly applying (3.27).

In total, this inversion method allows to compute derivatives of the volume rendering algorithm with arbitrarily many samples along the ray. The memory requirement is constant in the number of samples. In Paper D [WW22], both the Forward Differentiation and Adjoint Differentation are implemented to provide gradients for the camera parameters, step size of ray marching, the transfer function parameters and the volume densities. Based on how many parameters are needed, the more efficient algorithm is selected. For example, for optimizing only the orientation of the camera, Forward Differentiation is faster. For recovering the volume densities from images, where thousands of voxels need to be optimized, Adjoint Differentation is used.

We further compare in Paper D the reconstruction quality of different optimization algorithms for tomographic reconstruction. Here, an absorption-only volume is reconstructed from images. This allows us to compare against specialized methods from medical imaging, implemented in the ASTRA toolbox [van+15; Aar+16]. As a second baseline, the differentiable Monte Carlo path tracer Mitsuba 2 [ND+19] by Nimier-David *et al.*, is used. We show, that all three methods lead to similar results, except for a smoke plume dataset with a high absorption, where Mitsuba 2 fails to converge. Our proposed method outperforms both baseline methods in terms of the PSNR. Regarding performance, the specialized method for absorption-only reconstructed implemented in ASTRA achieves a volume reconstruction withing a minute. Our proposed method requires around 12 minutes and Mitsuba 2 runs multiple hours until convergence. The measurements were taken 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 GPU.

3.3 Neural Networks

After establishing how arbitrary functions with many inputs can be differentiated and thus optimized in Sec. 3.2, we now summarize the techniques of deep neural networks. These networks were employed in Paper A and Paper B to accelerate the rendering process. For a detailed introduction to neural networks, we refer to the book by Goodfellow *et al.* [GBC16] and introduce here only the most important concepts.

In the simplest form, neural networks are functions $f_\Theta : \mathbb{R}^n \to \mathbb{R}^m$ that approximate an arbitrary target function $f_{\text{gt}} : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ with n input dimensions and m output dimensions by optimizing the parameters $\Theta \in \mathbb{R}^k$. Ω denotes the space of valid inputs for the specific problem and can be any

subset of $\mathbb{R}^n$. To goal of training neural networks is finding the k parameters that lead to the best match between f_Θ and f_{gt}. For this purpose, let

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}, i = 1, ..., N \text{ with } \boldsymbol{x}_i \in \Omega, \boldsymbol{y}_i = f_{\text{gt}}(\boldsymbol{x}_i) \tag{3.28}$$

be N pairs of function inputs and expected, known outputs. This set is called the *training set* in *supervised training*. Then we can write the training process as the minimization problem

$$\Theta = \arg\min_\Theta \left\{ \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\Theta(\boldsymbol{x}_i), \boldsymbol{y}_i) \right\}. \tag{3.29}$$

The function $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is called the *loss function* and measures the distance between the network prediction and the target value. After training, the hope is that the network learned a compact representation of the – possibly complex – target function f_{gt} in the latent space of k parameters and that the network produces sensible outputs for new $x \in \Omega$ not from the training set. The latter describes the *generalization* ability of neural networks to perform inference on novel data.

The simplest choices for the loss function include the mean squared error (MSE) $\mathcal{L}_{\text{MSE}}(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2^2$ or the mean absolute error (MAE, also called L_1-loss) $\mathcal{L}_{\text{MAE}}(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_1$. Apart from those two loss functions, many further loss functions have been developed for specialized applications. This includes a smoothed version of the L_1-loss [Gir15] as a variation of the L_1-loss to increase training stability, or the cross entropy often used in classification tasks [RFB15]. If the output space represents images, see Sec. 3.3.2, special loss functions tailored for images can be used, e.g. the structure similarity index measure (SSIM) [Wan+04] or the learned perceptual image patch similarity (LPIPS) [Zha+18].

To train the networks, gradient-based approaches are typically employed. Using backpropagation (the Adjoint method), see Sec. 3.2.2, the gradients $\hat{\Theta} := \partial \mathcal{L} / \partial \Theta$ are computed. Then, gradient descent

$$\Theta_{i+1} = \Theta_i - \alpha \hat{\Theta}_i \tag{3.30}$$

with a step size or *learning rate* of α is employed to incrementally approach a local optimum. Note the abuse of notation: α denotes now the learning rate following the notation by Kingma and Ba [KB15], not the opacity as used in Sec. 3.1.2. In practical applications, derivatives for the whole training set cannot be computed at once due to memory limitations. Therefore, batches of the training data are processed one after another, leading to *stochastic gradient descent* (SGD). To improve the stability of the optimization process, more advanced optimization routines were developed, including SGD with momentum [Sut+13], Adam [KB15], Adagrad [DHS11], Adadelta [Zei12], or approximations of the second derivative (L-BFGS [NW99; Byr+16]).

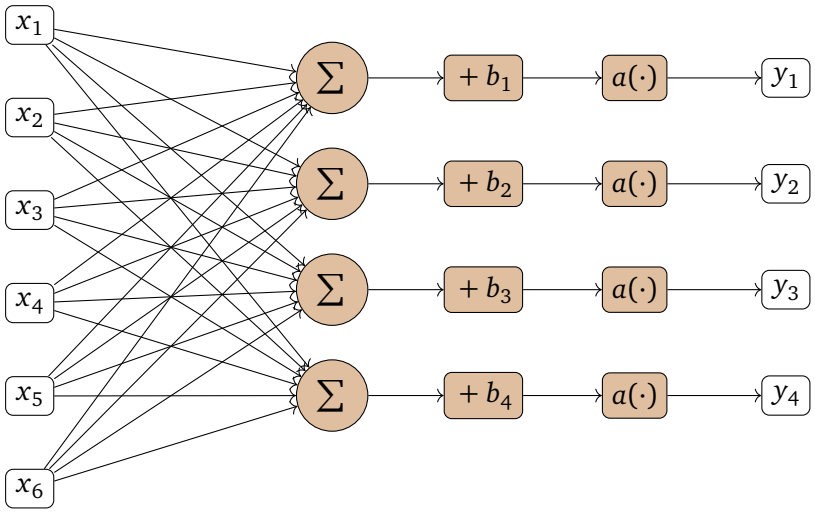


Figure 3.9: Visualization of a mulitlayer perceptron with $n_{\text{in}} = 6$ input and $n_{\text{out}} = 4$ output features as a network of neurons.

3.3.1 Fully-connected Neural Networks

Fully-connected neural networks, sometimes called feedforward networks or multilayer perceptrons (MLPs) [GBC16], are one of the oldest and simplest form of neural networks. Here, the network is assembled out of a sequential sequence of layers f_i of the following form:

$$f_i(\boldsymbol{x}) := a_i\left(W_i^T \boldsymbol{x} + \boldsymbol{b}_i\right), \tag{3.31}$$

where $W_i \in \mathbb{R}^{n_{\text{in}} \times n_{\text{out}}}$ is the *weight matrix*, $\boldsymbol{b}_i \in \mathbb{R}^{n_{\text{out}}}$ is the *bias vector* and $a_i : R^{n_{\text{out}}} \to \mathbb{R}^{n_{\text{out}}}$ is the *activation function* that is typically applied element-wise. Then the final network is a concatenation of K individual layers, $f_\Theta := f_K \circ ... \circ f_2 \circ f_1$ and the weight matrices and bias vectors are collected in the parameters $\Theta = \{W_1, b_1, ..., W_K, b_K\}$. Note that for the layers to fit together, n_{out} of layer i must match n_{in} of layer $i+1$. Otherwise, the layer sizes $n_{\text{in}}, n_{\text{out}}$, as well as the number of layers K, are discrete hyperparameters that have to be tweaked manually to achieve the desired performance in terms of evaluation speed and approximation quality. For an introduction to how hyperparameter tuning can be automated, we refer to Feurer and Hutter [FH19].

An MLP can be interpreted as a set of "neurons", where each neuron of the current layer computes a weighted sum of the activation of all neurons from the previous layer. This is then followed by adding a bias value and applying the non-linear activation function per neuron. A visualization of this interpretation can be found in Fig. 3.9.

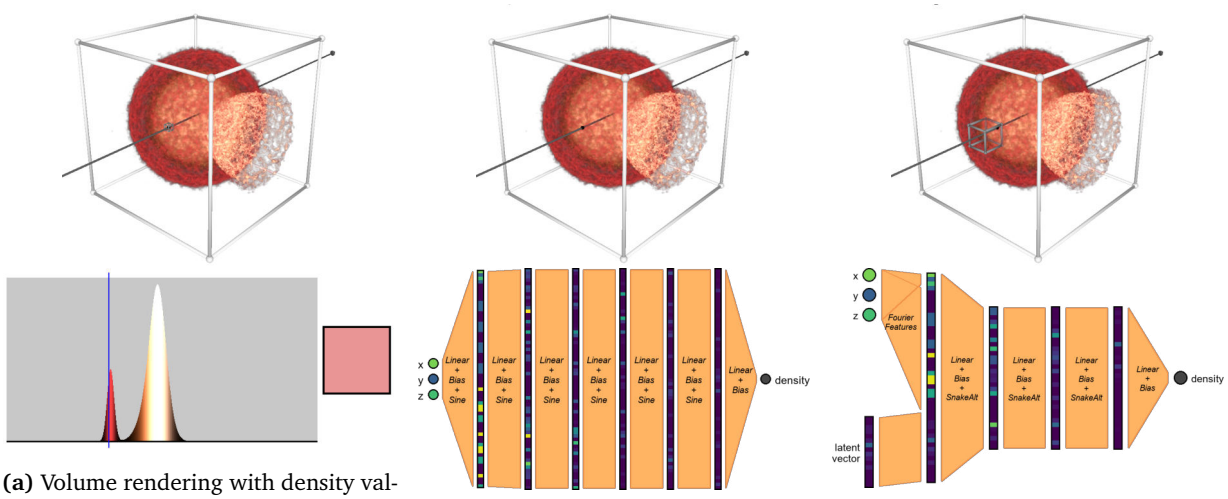
A central design decision when developing MLPs is the choice of the activation function $a(\cdot)$. The three basic activation functions used most early are the Sigmoid (σ), hyperbolic tangent (tanh) and rectifier linear unit (ReLU) [Jar+09; NH10], defined as

$$a_\sigma(x) = \text{sig}(x) := \frac{1}{1+\exp(-x)}\,, \quad a_{\text{tanh}}(x) := \tanh(x)\,, \quad a_{\text{ReLU}}(x) := \max\{0, x\}. \tag{3.32}$$

Goodfellow *et al.* [GBC16] recommend using a *ReLU* activation function as the default choice. Over the years, many variations of the ReLU-activation functions have been proposed [CUH16; How+17; Xu+15; He+15; Kla+17]. A concrete example is LeakyReLU proposed by Maas *et al.* [MHN+13]. It is defined as $a_{\text{leaky}}(x) := \max\{0, x\} + s\min\{0, x\}$ and replaces the hard zero in the negative regions by a flatter slope with angle $s = 0.01$. The authors show that this variation of ReLU helps in training the networks as zero gradients in regions of negative values are avoided.

In recent years, *Scene Representation Networks* (SRNs) emerged as an application of fully-connected networks. These are networks that map from a position in $\mathbb{R}^3$ with an optional direction to density or color. SRNs provide an implicit, compact representation of a 3D object and replace the volume V stored on a grid from Sec. 3.1. They are trained either from the 3D objects themselves [Mes+19; Mar+21; Tak+21; DNJ20; LJM21; Cha+20; Lu+21] or from 2D images for the task of volume reconstruction or novel-view synthesis [SZW19; Mil+20; Tan+20; Bar+21; Gar+21; Pum+21]. Later works improved the performance by replacing parts of the method by grids [Hed+21; Yu+21], estimating the depth of the hit [Nef+21], pre-integrating ray segments [LMW21], or caching of already-evaluated samples [Gar+21]. Further applications of SRNs include neural encodings of bidirectional reflectance distribution functions (BRDFs) [Szt+21], bidirectional texture functions (BTFs) [Rai+20] or volumetric texture patches (NeRF-Tex) [Baa+21].

Lu *et al.* [Lu+21] show, that SRNs can successfully compress 3D volumes, see Fig. 3.10b. The SRNs are trained on samples of (position, density) and encode the density volume in the weights of the network. The compression rate and quality is controlled by the size of the network. In unpublished work [WHW21], we show that the training time can be improved by up to 9× and the rendering performance by over 100× with the following two changes. First, the majority of the trainable parameters are moved to a coarse latent grid. This grid is trained jointly with the network. Interpolated latent vectors are then passed on to a much smaller network, we use only 4 layers with 32 channels, and converted to density output. With this, much fewer operations are needed to evaluate a position in space and thus speedups the training and inference. The quality of the compression is further no longer controlled by the size of the network, but instead by the size of this latent grid. Second, a further speedup in the inference is achieved by using a custom CUDA kernel for the network evaluation. In traditional deep learning frameworks like PyTorch [Pas+19] or TensorFlow [Aba+16], every network layer first reads the inputs from global memory, processes the values, and writes the



(a) Volume rendering with density values sampled from a grid and then transformed by the transfer function.

(b) Volume rendering with density values fetched directly from a network that encodes all information in the network weights.

(c) Volume rendering with a learned coarse latent grid. The interpolated latent vectors are sent to a smaller network and converted to density samples.

Figure 3.10: Volume rendering with scene representation networks (SRN). In traditional volume rendering (a), the volume densities are fetched from a fine grid. (b) Lu *et al.* [Lu+21] show that high compression rates with good quality can be achieved by encoding the whole volume in the network weights of an SRN. (c) We show that the training and rendering performance can be significantly improved by transferring the majority of the parameters to a learnable, coarse latent grid and interpreting those values with a smaller network. Images inspired by Fig. 2 of Weiss *et al.* [WHW21].

results back to memory. Especially for small networks, this introduces a large memory overhead. We show, how this memory overhead can be avoided by keeping the intermediate states between network layers in fast local memory. Concurrent work by Müller *et al.* [M+21; M+22] propose a similar custom inference for the tasks of novel view synthesis and radiance caching, among others. The authors report similar performance improvements.

3.3.2 Convolutional Neural Networks

The predecessor of convolutional neural networks was introduced by Kunihiko Fukushima [Fuk80] in 1980, called the “Neocognitron”. The main motivation is to design a neural network that mimics the visual cortex: the ability to recognize patterns unaffected by a shift in position. The neurons, called “cells” by Fukushima, are aligned on a 2D grid the inputs of those neurons all follow the same spatial pattern. Between different cells, only the positions are shifted, see Fig. 3.11. Further early works include Le Cun *et al.* [LeC+89; LC+89], that extend upon the work above for the task of handwritten digit recognition, and Weng *et al.* [WAH93] that introduce *max pooling*.

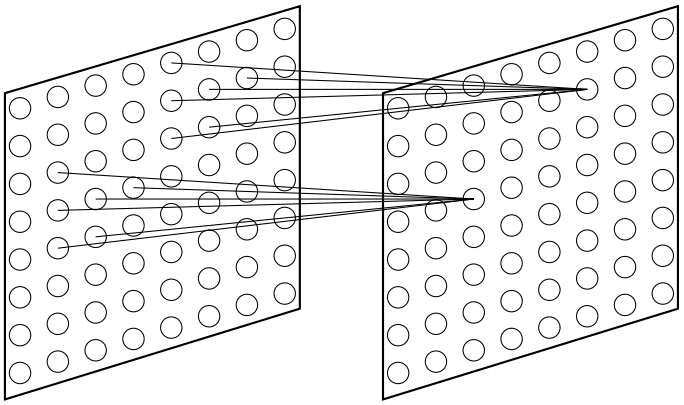


Figure 3.11: Early convolutional neural networks: Neural network where the input neurons follow the same spatial pattern. Image inspired by Fukushima [Fuk80].

Let I be a two-dimensional input image of size $H \times W$ and S the two-dimensional output image. Let K be the two-dimensional filter kernel with $M \times N$ entries. Then the output S is computed using the *cross-correlation* defined as [GBC16]

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{3.33}$$

Typical deep learning frameworks like PyTorch [Pas+19] or TensorFlow [Aba+16] implement the aforementioned cross-correlation operation instead of the convolution operation for its more natural interpretation. The cross-correlation is identical to a convolution if the kernel K is flipped. Common kernel sizes are 3×3, 5×5 or 7×7, [LC+89; WAH93], but larger kernels have been used. For example, Krizhevsky *et al.* [KSH12] use a kernel of size 11×11 in the first layer of their network architecture.

In typical applications, the input does not contain a single channel, but rather L channels, i.e. $I \in \mathbb{R}^{H \times W \times L}$, a three-dimensional tensor. Similarly, the output image S contains O channels, i.e. $O \in \mathbb{R}^{H \times W \times O}$. Then, the filter kernel combines all input channels and output channels, $K \in \mathbb{R}^{M \times N \times L \times O}$, a four-dimensional tensor. The updated cross-correlation is defined as

$$S(i,j,o) = (K * I)(i,j,o) = \sum_m \sum_n \sum_l I(i+m, j+n, l) K(m,n,l,o). \tag{3.34}$$

One application area of convolutional neural networks are image classification tasks [Jar+09; KSH12; RFB15; Lee+09]. Large standarized datasets like the ImageNet database [Rus+15] allow to train large network architectures and also allow for reproducible results and comparisons between different architectures. To reduce images with a size of, e.g., $224 \times 224 = 50,176$ pixels down to 1000 classes in the ImageNet database, the resolution of the images have to be reduced. One way to reduce the

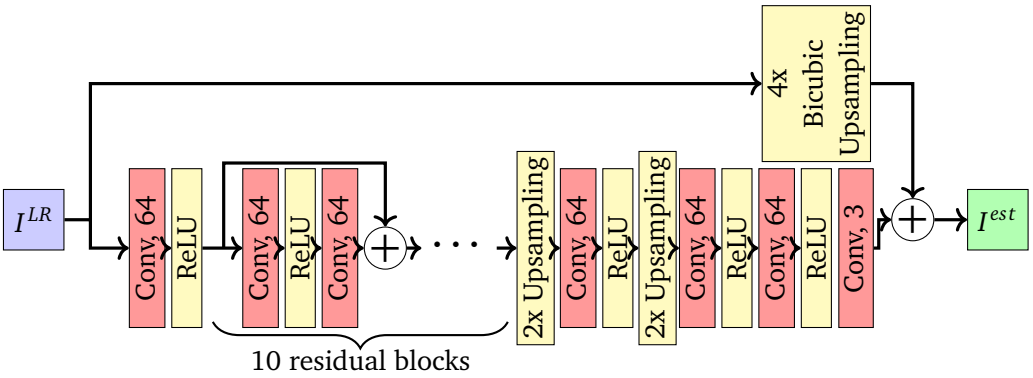


Figure 3.12: 4× super-resolution of three-channel RGB-images using the EnhanceNet architecture by Sajjadi *et al.* [SSH17]. First, a convolutional layer pre-processes the color inputs and produces an image with 64 channels. Then, ten so-called residual blocks with two convolutional layers each that learn changes to their inputs are used. To upscale the image, two 2× nearest-neighbor upsampling layers are used, followed by another convolutional layers. The whole network learns the residual between a bicubic interpolation of the input and the ground truth. All convolutional layers use 3×3 kernels. Image inspired by Fig. 4 of Weiss *et al.* [Wei+21].

image resolution is using pooling operations. More specifically, *max pooling* [WAH93] takes blocks of size h, w and replaces them with the maximal value of that block,

$$S(i,j) = \max_{i'=0,\dots,h-1} \max_{j'=0,\dots,w-1} I(hi+i', wj+j'). \tag{3.35}$$

This effectively reduces the size of a input image from resolution hH, wW to the output resolution H, W. Another method to reduce the resolution is the use of strided convolutions [GBC16],

$$S(i,j,o) = c(K,I,s)_{ijo} = \sum_m \sum_n \sum_l I(is+m, js+n, l) K(m,n,l,o), \tag{3.36}$$

where the stride s defines how many pixels are skipped in the input image between the pixels of the output image. As an example, Krizhevsky *et al.* [KSH12] use both max pooling and strided convolutions in their architecture. For an extended description of the arithmetic behind the various convolution operations, how to handle the border, and visual explanations, we refer to the article by Dumoulin and Visin [DV16].

Another application area is image super-resolution. Here, the task is to recover a high-resolution image from a low-resolution image. One concrete example is the EnhanceNet architecture by Sajjadi *et al.* [SSH17], sketched in Fig. 3.12. The EnhanceNet is designed to perform a 4× super-resolution by learning the difference between a bicubic upsampling of the input image and a target ground-truth image. For a discussion of other architectures, we refer to Sec. 2.2.

3.3.3 Isosurface Super-Resolution

We employ the EnhanceNet architecture for image super-resolution in Paper A [Wei+21] to upsample isosurface renderings at low screen resolutions. By rendering only a low-resolution version of the object and recovering the high-resolution image using a network, the rendering process can be accelerated. We show, that it is beneficial for the reconstruction quality to upsample the normal map with supervised losses on the normal map, instead of performing super-resolution on the shaded RGB color image. Only during inference, a screen-space shading is applied to compute the color image from the normal map. This way, the network is decoupled from lighting and shading, i.e. different settings for the strength of specular reflections, and can focus better on reconstructing the geometry. To help the network perform the inference, the depth map and a binary mask stating whether the ray under the current pixel hit the object are passed to the network as further input features.

Additionally, we show, that the super-resolution network can learn to produce an ambient occlusion (AO) map as an additional output jointly to the other features. This AO map is trained in a supervised fashion given a ground truth AO map, as done with the normal map as well. Ambient occlusion describes the shadowing in cavities as an approximation of global illumination. This helps the user to interpret the 3D structure. The whole pipeline including AO estimation is visualized in Fig. 3.13. In a related work, Engel and Ropinski [ER21] use 3D convolutions to predict a volumetric AO volume from the density volume and transfer function that is then directly used in the raytracing process. They especially demonstrate and compare various strategies how the TF can be incorporated into the network.

Since the network is tasked to "hallucinate" new features in the high-resolution images, temporal coherence becomes a problem. Without a form of temporal coherence, the predictions are unstable and lead to distracting flickering artifacts during interactive explorations, e.g., when the camera is changed. To compensate that, we apply the frame-recurrent video super-resolution architecture (FRVSR) by Sajjadi *et al.* [SVB18]. Here, the prediction of the previous frame is warped by the optical flow and added as an extra input to the network. A temporal loss during training then ensures, that the network generates features that are consistent with the previous frame. The optical flow describes the motion of the observed objects from one image to another. It can be computed as a side-product of the isosurface renderer. Let $p = (i, j)$ be the current pixel coordinates and $\boldsymbol{x} \in \mathbb{R}^3$ be the point in 3D where the isosurface intersection at that pixel was detected. Let $C^{(t)}$ be the camera transformation for the current frame, i.e, $p = C^{(t)}(\boldsymbol{x})$. During rendering of the current frame, the previous camera matrix $C^{(t-1)}$ is known. Then the optical flow is given by

$$\Delta p = C^{(t)}(\boldsymbol{x}) - C^{(t-1)}(\boldsymbol{x}). \tag{3.37}$$

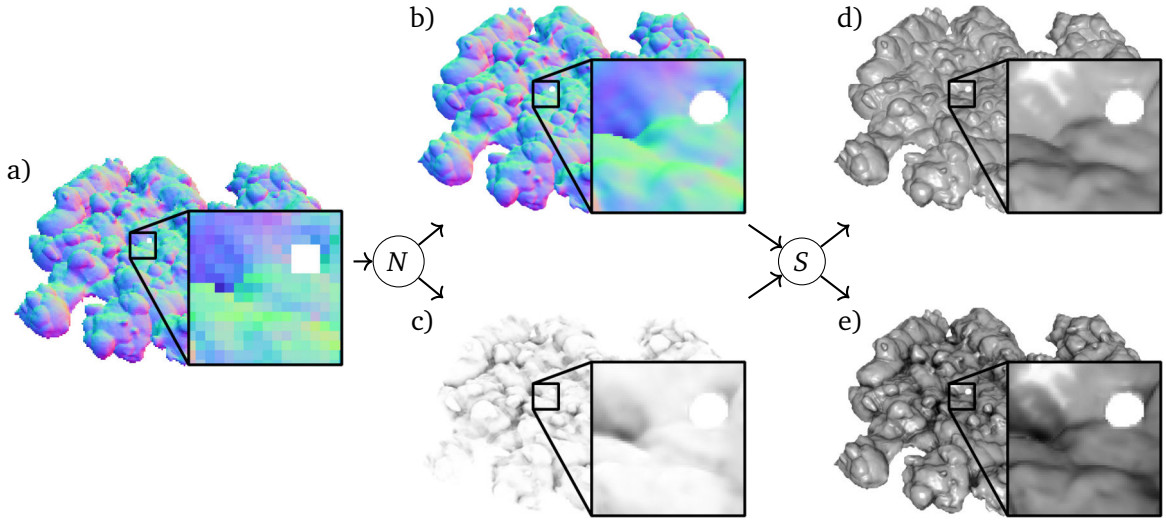


Figure 3.13: Visualization of the isosurface super-resolution pipeline [Wei+21] for inference during rendering. (a) A low-resolution normal map is rendered. The super-resolution network N then predicts the (b) high-resolution normal map and (c) high-resolution ambient occlusion (AO) estimation. A screen space shading step S then computes the final color, either (d) without AO or (e) with AO, here exaggerated for demonstration purpose. Additional depth maps and masks as network input and output are omitted for clarity of visualization. Cloud dataset courtesy of Kallweit *et al.* [Kal+17].

The optical flow is then used to align the previous frame with the current frame. This method allows to effectively reduce the temporal flickering during camera motion. A limitation, however, is, that only camera movements are detected and handled by the optical flow. Changes to the object itself, i.e. changing the isovalue, are not detected. To support changes to the object as well, the optical flow would have to be estimated from the images directly, i.e. via optimization [DN11] or neural networks [Dos+15; Ilg+17] in future work.

3.3.4 Adaptive Sampling

In the above super-resolution approach [Wei+21], the image was upsampled with a factor of 4×. This can be interpreted as a regular sampling where only every 4^{th} pixel in every dimension is rendered. This implicitly assumes, however, that every pixel is equally important. We propose in Paper B [Wei+20] an adaptive sampling that achieves better quality with the same number of rendered pixels by placing those samples in more important regions. This sampling is performed data-driven, i.e. learned to be optimal for a given set of training data and a trainable reconstruction.

The stages of the proposed pipeline are shown in Fig. 3.14 through the inputs and outputs of each stage. First, a low-resolution image L with, e.g., an 8^{th} of the resolution is rendered. This low-resolution input informs a first neural network called the *importance network* of where features

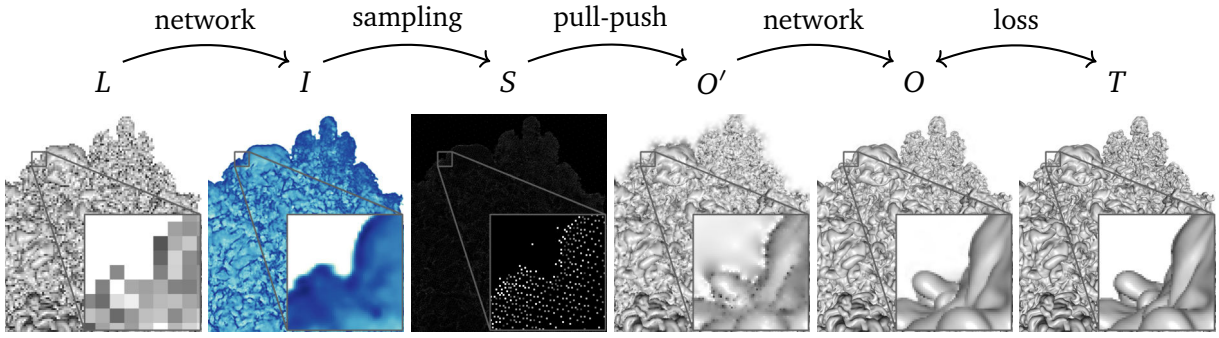


Figure 3.14: Illustration of the adaptive sampling pipeline [Wei+20]. Starting with a low-resolution image L, an importance network estimates the importance map I. A differentiable sampler then renders rays at sparse locations S. A first interpolation using the pull-push algorithm [Kra09] O' prepares the image for refinement performed by a reconstruction network that predicts the final output O. A supervised loss compares this prediction to the target image T.

might be located. We use a variation of the EnhanceNet (see Fig. 3.12) with only five residual blocks and an extra upsampling layer at the end to reach the 8× upsampling for this task. The output of this stage is an importance map I containing the non-negative importance value per pixel. It is visualized in Fig. 3.14 via a colormap for better visibility. A differentiable sampling stage, detailed below, then renders the desired number of samples using raytracing, resulting in the sparsely filled image S. To reconstruct the final image, a *reconstruction network*, an EnhanceNet without the final upsampling layers, is used. We found, that using a baseline inpainting step that already provides a good estimate of the interpolated image from the samples improves the reconstruction O'. A differentiable version of the pull-push algorithm [Kra09] is used for this purpose. The reconstruction network then only has to learn how to improve and sharpen the image, resulting in the output O. The whole pipeline is trained end-to-end using a supervised loss on the final reconstructed image against a target image T. The importance network is never explicitly told where to place the samples, it is trained using gradients propagated through the reconstruction network and the sampling stage.

One of the core contributions of this work is the definition of a differentiable sampling stage. In related work that performs adaptive sampling for Monte Carlo rendering, the task is the specify between one to many samples per pixel [KKR18; Has+20]. For direct volume rendering, however, each pixel is already noise-free as global illumination effects are ignored. Therefore, the task is to sample between zero to one sample per pixel, i.e. a sparse sampling of the entire image.

Let $I \in \mathbb{R}^{H \times W}$ be the importance map estimated by the importance network with mean μ_I. First, the importance map is normalized to a certain user-defined mean μ and a small minimal value l. By controlling the target mean, the user can define how many pixels to sample. If, for example, 5% of rays should be sampled, μ is set to 0.05. The minimal value serves as a lower bound to sample at least

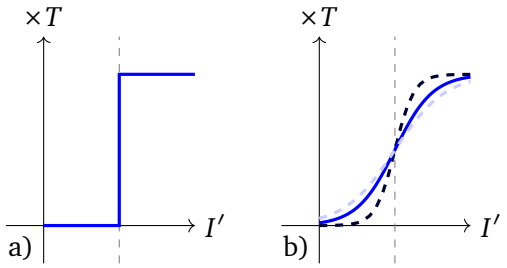


Figure 3.15: Visualization of the smooth approximation to make sampling differentiable. a) During inference, the selection of the pixel can be interpreted as a step function multiplied on the target image T. b) For training, this step function is approximated with a sigmoid function of various steepness.

a few pixels in otherwise empty regions. This helps later with the reconstruction. The new importance map is then defined as

$$I' := \min\left\{1, l + I\frac{\mu - l}{\mu_I + \epsilon}\right\}, \tag{3.38}$$

with a small $\epsilon > 0$ to avoid division by zero in case of an empty image. Next, let $P \in [0,1]^{H\times W}$ be a sampling pattern that provides a pseudo-random number per pixel. We found that a low-discrepancy sampler like Plastic sampling [Rob20] performs better than uniform sampling. During inference, a pixel i, j is sampled and rendered if $I'_{ij} > P_{ij}$, resulting in the sparsely sampled image S. This rejection sampling, however, is not differentiable. Therefore, it cannot be used to propagate gradients to the importance network. For training, two changes are applied.

First, the rejection sampling is replaced by a step function multiplied by the target high-resolution image T,

$$S_{ij} := \mathbb{1}_{I'_{ij} - P_{ij}} T_{ij}, \tag{3.39}$$

illustrated in Fig. 3.15a. The target image T is required for the supervised loss on the output and therefore already available. This way, the renderer does not need to be included in the training routine, increasing the training performance. Second, to make (3.39) differentiable for training, it is approximated by a smooth sigmoid function, see (3.32),

$$S_{ij} := \mathrm{sig}\Big(\alpha(I'_{ij} - P_{ij})\Big)\, T_{ij} \tag{3.40}$$

The hyperparameter α, visualized in Fig. 3.15b, influences the steepness of the sigmoid approximation. We found a value of $\alpha = 50$ to be optimal [Wei+20]. A smaller value leads to decreased inference performance, as the discrepancy between the sigmoid approximation and the hard rejection sampling during testing increases. For larger values of α, the training becomes unstable due to vanishing gradients. With this pipeline, a neural network can learn to place samples in the locations required for a good reconstruction.

The proposed adaptive sampling pipeline is trained on only a few datasets, a few timesteps of the Ejecta supernova simulation, see Fig. 3.1 for an example rendering. Regardless, the networks generalize well to novel, unseen datasets. Unlike in the work on isosurface super-resolution (see Sec. 3.3.3), we demonstrate, that the adaptive sampling pipeline can also be applied for direct volume renderings [Wei+20]. In this case, the reconstruction network directly predicts the color output, instead of the normal map combined with a screen-space shading as in the case of isosurfaces. To handle the different color schemes introduced when the user designs a custom transfer function, random transfer functions are generated and used during training.

4

Paper A: Volumetric Isosurface Rendering with Deep Learning-Based Super-Resolution[1]

Abstract Rendering an accurate image of an isosurface in a volumetric field typically requires large numbers of data samples. Reducing this number lies at the core of research in volume rendering. With the advent of deep learning networks, a number of architectures have been proposed recently to infer missing samples in multidimensional fields, for applications such as image super-resolution. In this article, we investigate the use of such architectures for learning the upscaling of a low resolution sampling of an isosurface to a higher resolution, with reconstruction of spatial detail and shading. We introduce a fully convolutional neural network, to learn a latent representation generating smooth, edge-aware depth and normal fields as well as ambient occlusions from a low resolution depth and normal field. By adding a frame-to-frame motion loss into the learning stage, upscaling can consider temporal variations and achieves improved frame-to-frame coherence. We assess the quality of inferred results and compare it to bi-linear and cubic upscaling. We do this for isosurfaces which were never seen during training, and investigate the improvements when the network can train on the same or similar isosurfaces. We discuss remote visualization and foveated rendering as potential applications.

Contribution The method development and implementation was done by the first author. Parts of the neural network architecture were developed and provided by Mengyu Chu. Discussions with Rüdiger Westermann, Nils Thuerey, and Mengyu Chu led to the final paper.

IEEE Transactions on Visualization and Computer Graphics, Volume 27, Issue 6, June 2021, pp. 3064 – 3078

5

Paper B: Learning Adaptive Sampling and Reconstruction for Volume Visualization[1]

Abstract A central challenge in data visualization is to understand which data samples are required to generate an image of a data set in which the relevant information is encoded. In this work, we make a first step towards answering the question of whether an artificial neural network can predict where to sample the data with higher or lower density, by learning of correspondences between the data, the sampling patterns and the generated images. We introduce a novel neural rendering pipeline, which is trained end-to-end to generate a sparse adaptive sampling structure from a given low-resolution input image, and reconstructs a high-resolution image from the sparse set of samples. For the first time, to the best of our knowledge, we demonstrate that the selection of structures that are relevant for the final visual representation can be jointly learned together with the reconstruction of this representation from these structures. Therefore, we introduce differentiable sampling and reconstruction stages, which can leverage back-propagation based on supervised losses solely on the final image. We shed light on the adaptive sampling patterns generated by the network pipeline and analyze its use for volume visualization including isosurface and direct volume rendering.

Contribution The method development and implementation was done by the first author. Automatic generation of transfer functions for training was developed by Mustafa Işık. Discussions with Rüdiger Westermann and Justus Thies led to the final paper.

IEEE Transactions on Visualization and Computer Graphics, early access

6

Paper C: Analytic Ray Splitting for Controlled Precision DVR[1]

Abstract For direct volume rendering of post-classified data, we propose an algorithm that analytically splits a ray through a cubical cell at the control points of a piecewise-polynomial transfer function. This splitting generates segments over which the variation of the optical properties is described by piecewise cubic functions. This allows using numerical quadrature rules with controlled precision to obtain an approximation with prescribed error bounds. The proposed splitting scheme can be used to find all piecewise linear or monotonic segments along a ray, and it can thus be used to improve the accuracy of direct volume rendering, scale-invariant volume rendering, and multi-isosurface rendering.

Contribution The method development and implementation was done by the first author. Discussions with Rüdiger Westermann led to the final paper.

[1]

7

Paper D: Differentiable Direct Volume Rendering[1]

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.

Contribution The method development and implementation was done by the first author. Discussions with Rüdiger Westermann led to the final paper.

 IEEE Transactions on Visualization and Computer Graphics, Volume 28, Issue 1, January 2022, pp. 562 – 572

8 Final Discussion

In this thesis, we have proposed methods to integrate neural networks into the visualization pipeline, to use controlled-precision numerical schemes for direct volume rendering, and to make volume rendering itself differentiable. This allows accelerating the rendering during interactive exploration tasks, reducing rendering artifacts, and inferring features of the underlying data – camera parameters, transfer functions, and volume densities – from images.

8.1 Future Work

In Paper A [Wei+21] and Paper B [Wei+20], we have shown, how neural networks can perform fixed or adaptive super-resolution to improve the rendering performance. These networks, however, worked sorely on images. The rays that are rendered all use the same step size and, unless auxiliary acceleration structures as presented in Sec. 2.1 are used, traverse a lot of empty or uniform areas. Therefore, in future research, extending the adaptivity in image-space to an adaptivity along the sampled ray could be investigated. Mildenhall *et al.* [Mil+20] trained a scene representation network (SRN) evaluated with a coarse step size that predicts a piecewise constant PDF of the absorption along the ray. This PDF is then used to sample the locations for the evaluation of the final "fine" network that predicts the colors. This first "coarse" network, however, was trained together with the "fine" SRN to represent a specific scene and, thus, does not generalize to novel scenes. Furthermore, the volume data is typically given in a visualization application and sampling such a volume is comparably cheap. Designing a network that is fast enough to compare to classical volume sampling when evaluated at every sample location, is challenging. Instead, one could approach this challenge in the following way: In the proposed adaptive sampling pipeline [Wei+20], the first importance network does not only predict where to sample, but also – with the help of additional auxiliary features from the low-resolution rendering – what the expected distribution of samples along the ray looks like. During the high-resolution rendering,

the step size is then dynamically adjusted based on the predicted analytical distribution, coarser in uniform areas and finer in highly detailed areas. How to model such a distribution and how to train the network is still an open question.

In a similar context, but a different research question is the acceleration of the controlled-precision quadrature scheme presented in Paper C [WW21]. This method involves voxel traversal, isosurface intersections, polynomial multiplication, and numerical quadrature, and is computationally intensive. Improving the performance of this method, therefore, is desired. One possibility would be to investigate a hybrid method. For smooth, uniform regions, constant stepping already provides good solutions. For highly detailed areas with sharp peaks in the TF, a small step size would be needed to still achieve good quality with constant stepping. In these regions, a hybrid algorithm could switch to the proposed controlled-precision quadrature scheme. As a further step, a combination with a network that predicts the step size or a sample distribution, see above, could be used to split the rays into segments processed by constant stepping and by controlled-precision quadrature in advance.

Back to network-based super-resolution methods, those methods are tasked with hallucinating new features. This is inherently unstable over time. In a first work [Wei+21] we have shown how to reduce inconsistencies over short time spans by passing the previous prediction as input to the network. This effectively reduces high-frequent flickering, but low-frequency inconsistencies over time remain. In the future, reducing also these errors over longer time sequences is desirable. This could be solved, e.g., by incorporating a history of multiple frames from time sequences at different frame rates into the prediction [Iso+20].

For differentiable volume rendering (Paper D [WW22]), we have shown several synthetic examples on optimization tasks (camera) and reconstruction tasks (TF and volume). As a first concrete application, the proposed differentiable renderer was applied to train scene representation networks (SRNs) from images [WHW21]. One open question remains, how to automatically generate and sample transfer functions that carry semantically meaningful information. We have yet to see a metric, ideally differentiable, that measures if an image generated by a specific transfer function is “meaningful”. The difficulty lies in the definition of a “meaningful” visualization, as this highly depends on the use case and target audience. Hence, most works on TF generation use a semi-automatic process to aid an expert in designing TFs [CM10; Rui+11]. As a possible approach to this problem, databases of visualizations generated by experts were used to measure the quality of a viewpoint and to sample good camera positions [Tao+16; ST19; Yan+19a] using feature matching and voting systems. In future work, we want to investigate whether such matching and voting systems, or other approaches like style-transfer [GEB16; JAFF16] as image losses, can be used to optimize semantically plausible TFs through the proposed differentiable rendering system.

8.2 Conclusion

We started this thesis with the task of accelerating volume visualizations. In a first work [Wei+21] we proposed to use a neural network developed for image and video super-resolution [SSH17; SVB18] to upsample images of isosurfaces rendered to a low-resolution screen. This allows reducing the number of rays that have to be rendered and improves the rendering performance on large datasets when ray traversal becomes expensive. While the original network is applied to RGB-images, we found that the performance of the network is improved, if depth and normal maps are used instead. The final color is calculated in a screen-space shading step, which also has the benefit of decoupling light information from the network. To incorporate temporal consistency, we applied the technique by Sajjadi *et al.* [SVB18]. Here, the prediction of the previous frame is warped by the optical flow and passed additional input to the network. A temporal loss during training then forces the network to predict features that are consistent with the previous predictions. As a further improvement, the super-resolution network learns to predict the influence of ambient occlusion per pixel which is then applied to the image in the shading step. Ambient occlusion greatly improves the depth perception of the image. To compensate for possible reconstruction errors introduced by the network, we presented foveated rendering [Gue+12] as one possible application. During foveated rendering, the high-resolution image is rendered in the focus region where errors introduced by the network would be noticeable the most. In the periphery, a low-resolution image is rendered to save computation time and then upsampled by the network to hide the transition between low- and high-resolution areas.

The above method assumed that all pixels are equally important to motivate a fixed super-resolution method of, e.g., 4×. To improve the quality while keeping the number of sampled pixels constant, we next proposed an adaptive sampling pipeline [Wei+20]. Given a very low-resolution rendering as context, a first neural network estimates an importance map where important features are located. This importance map is then used to sample a user-defined number of pixels and compute the pixel values using raytracing. An inpainting step and a second neural network then reconstruct the final output. This whole pipeline is trained end-to-end with supervised losses only on the output image. In other words, the first importance network is never explicitly told where to place the samples, it is trained sorely on gradients from the reconstruction step. To the best of our knowledge, this is the first method that showed how to train an importance estimator purely based on what a second, learned reconstruction step requires for a sparse reconstruction. This indicates, that a neural network is capable of distinguishing important from unimportant regions. We have shown the application of the presented method for isosurface renderings and direct volume renderings. Trained only on a few datasets, the networks generalize well to unseen datasets and – in the case of DVR – to novel transfer functions. For large screen sizes and volume sizes, the presented pipeline outperforms a baseline rendering where all pixels are rendered.

The network-based super-resolution and reconstruction approaches can only provide an approximation and are prone to errors. The rendering itself, however, is also prone to numerical errors, especially when using direct volume rendering. The direct volume rendering integral cannot be solved analytically for the general case. Previous analytical or controlled-precision numerical schemes only exist for special cases [NA92; Boe+97; EKE01; Kni+03]. In the general case, raymarching with a constant step size is used, comparable to a quadrature via the rectangular rule. If the step size is chosen too large, artifacts in form of "ringing" (compare Fig. 3.7a) occur. Choosing the step size arbitrarily small is no alternative, because the computational cost scales linearly with the number of samples along the ray, i.e., inversely with the step size. We showed, that for a larger class of DVR scenarios, namely tri-linearly interpolated hexahedral grids with a piecewise cubic transfer function, controlled-precision solutions are available that guarantee that no features of the data highlighted by the TF are missed [WW21]. For this purpose, we observed that the control points of the piecewise TF define isosurfaces of the volume. With the numerical scheme by Marmitt *et al.* [Mar+04], those isosurfaces can be extracted exactly and the ray segmented along voxel boundaries and isosurfaces of the TF control points. For each such segment, the density interpolation, as well as the absorption and color defined by the TF are polynomial functions. This allowed us to evaluate the absorption integral analytically and the emission integral numerically with a controlled error bound based on the methods presented by Novins and Arvo [NA92]. We showed that the presented method can accurately render volume visualizations with high-frequent TFs and outperforms raymarching with a small constant step size in terms of speed and quality in those cases.

Finally, we presented, how to make the rendering process itself differentiable [WW22]. We discussed two techniques of automatic differentiation, Forward and Adjoint Differentiation, their strengths and weaknesses, and how they can be applied in the context of volume rendering. In Forward Differentiation, derivatives are propagated jointly to the regular program execution. This is implemented via operator overloading. This differentiation method scales linearly with the number of parameters to optimize for and is thus most effective if only a few parameters like the camera position are optimized. In Adjoint Differentiation, the derivatives are propagated in reverse order. This allows to compute derivatives for many parameters at once and is optimal for many-parameter optimizations like TF or volume density reconstruction. Adjoint Differentiation, however, requires storing the intermediate results for the gradient computations. To remedy this memory cost, we observed, that by analytically inverting specific operations in the direct volume rendering algorithm, namely the blending step, these intermediate results can be reconstructed and do not need to be stored. This way, gradients can be computed using Adjoint Differentiation with arbitrarily many operations, i.e., arbitrarily many steps along the ray. A similar idea of analytic inversion for differentiation was developed concurrently by Vicini *et al.* [VSJ21] for Monte-Carlo path tracing with multiple scattering events. We presented examples for optimizing the camera location based on entropy measures, reconstructing the transfer

function from images and a known volume, and tomographic reconstruction of an absorption-only volume from images. We showed, that for camera optimization, Forward Differentiation performs best, and if more parameters are optimized like in TF optimization, Adjoint Differentiation outperforms Forward Differentiation. For the case of tomographic reconstruction of a volume rendered only with absorption, we conducted comparisons against specialized algebraic methods that support only this specific optimization [van+15; Aar+16] and against Mitsuba 2 [ND+19] for a general differentiable path tracer. We show, that our method achieves the best peak signal-to-noise ratio (PSNR) on the three tested datasets and is faster than Mitsuba 2 by one to two orders of magnitude.

Bibliography

[Aar+16] 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". In: *Opt. Express* 24.22 (2016), pp. 25129–25147. DOI: 10.1364/OE.24.025129.

[Aba+16] 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)*. 2016, pp. 265–283.

[ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein generative adversarial networks". In: *International Conference on Machine Learning*. 2017, pp. 214–223.

[An+21] Y. An, H.-W. Shen, G. Shan, G. Li, and J. Liu. "STSRNet: Deep Joint Space-Time Super-Resolution for Vector Field Visualization". In: *IEEE Computer Graphics and Applications* 41.6 (2021), pp. 122–132. DOI: 10.1109/MCG.2021.3097555.

[App68] A. Appel. "Some Techniques for Shading Machine Renderings of Solids". In: *Proceedings of the AFIPS spring joint computer conference*. AFIPS '68 (Spring). ACM, 1968, pp. 37–45. DOI: 10.1145/1468075.1468082.

[AW87] J. Amanatides and A. Woo. "A fast voxel traversal algorithm for ray tracing". In: *Eurographics*. Vol. 87. 3. 1987, pp. 3–10.

[Baa+21] H. Baatz, J. Granskog, M. Papas, F. Rousselle, and J. Novák. "NeRF-Tex: Neural Reflectance Field Textures". In: *Computer Graphics Forum*. Wiley Online Library. 2021. DOI: 10.1111/cgf.14449.

[Bar+21] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: (2021), pp. 5855–5864.

[BB+00] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon. “Automatic differentiation of algorithms”. In: *Journal of Computational and Applied Mathematics* 124.1-2 (2000), pp. 171–190. DOI: 10.1016/S0377-0427(00)00422-2.

[BHP15] J. Beyer, M. Hadwiger, and H. Pfister. “State-of-the-Art in GPU-Based Large-Scale Volume Visualization”. In: *Computer Graphics Forum* 34.8 (2015), pp. 13–37. DOI: 10.1111/cgf.12605.

[Bi+20] S. Bi, Z. Xu, P. Srinivasan, B. Mildenhall, K. Sunkavalli, M. Hašan, Y. Hold-Geoffroy, D. Kriegman, and R. Ramamoorthi. “Neural Reflectance Fields for Appearance Acquisition”. In: (2020). DOI: 10.48550/ARXIV.2008.03824.

[BLL19] M. Berger, J. Li, and J. A. Levine. “A Generative Model for Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.4 (2019), pp. 1636–1650. DOI: 10.1109/TVCG.2018.2816059.

[BM98] 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. New York, NY, USA: ACM, 1998, pp. 299–309. DOI: 10.1145/280814.280924.

[BMB97] A. Brandt, J. Mann, and M. Brodski. “An $O(N^2 \log N)$ Multilevel Backprojection Method”. In: *Gauss Center Report WI/GC* 6 (1997).

[Boa+01] D. A. Boas, D. H. Brooks, E. L. Miller, C. A. DiMarzio, M. Kilmer, R. J. Gaudette, and Q. Zhang. “Imaging the body with diffuse optical tomography”. In: *IEEE signal processing magazine* 18.6 (2001), pp. 57–75.

[Boe+97] M. W. de Boer, A. Gröpl, J. Hesser, and R. Männer. “Reducing artifacts in volume rendering by higher order integration”. In: *8th Annual IEEE Conference on Visualization; Late Breaking Hot Topics* (1997), pp. 1–4.

[Bra+09] C. Braley, R. Hagan, Y. Cao, and D. Gračanin. “GPU Accelerated Isosurface Volume Rendering Using Depth-Based Coherence”. In: *ACM SIGGRAPH ASIA 2009 Posters*. SIGGRAPH ASIA ’09. Yokohama, Japan: ACM, 2009. DOI: 10.1145/1666778.1666820.

[Bri+21] K. M. Briedis, A. Djelouah, M. Meyer, I. McGonigal, M. Gross, and C. Schroers. “Neural frame interpolation for rendered content”. In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–13.

[BRLP19] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. “TTHRESH: Tensor compression for multidimensional visual data”. In: *IEEE transactions on visualization and computer graphics* 26.9 (2019), pp. 2891–2903.

[Bru+19] V. Bruder, C. Schulz, R. Bauer, S. Frey, D. Weiskopf, and T. Ertl. “Voronoi-Based Foveated Volume Rendering”. In: *Computer graphics forum*. Wiley Online Library. The Eurographics Association, 2019.

[BS05] U. Bordoloi and H.-W. Shen. “View selection for volume rendering”. In: *IEEE Visualization (VIS)*. 2005, pp. 487–494. DOI: 10.1109/VISUAL.2005.1532833.

[BS08] L. Bavoil and M. Sainz. *Screen space ambient occlusion*. https://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf. Accessed: 2022-05-11. 2008.

[Byr+16] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. “A stochastic quasi-Newton method for large-scale optimization”. In: *SIAM Journal on Optimization* 26.2 (2016), pp. 1008–1031.

[CBE20] V. Careil, M. Billeter, and E. Eisemann. “Interactively modifying compressed sparse voxel representations”. In: *Computer Graphics Forum*. Vol. 39. 2. Wiley Online Library. 2020, pp. 111–119.

[CCF15] 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*. IEEE. 2015, pp. 17–24.

[CCL92] T. Chen, H. Chen, and R.-w. Liu. “A constructive proof and an extension of Cybenko’s approximation theorem”. In: *Computing science and statistics*. Springer, 1992, pp. 163–168.

[Cha+17] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. “Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–12.

[Cha+20] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe. “Deep local shapes: Learning local sdf priors for detailed 3d reconstruction”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 608–625.

[Che+16] T. Chen, B. Xu, C. Zhang, and C. Guestrin. “Training Deep Nets with Sublinear Memory Cost”. In: *arXiv preprint* (2016). DOI: 10.48550/ARXIV.1604.06174.

[Che+18] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. “Deep Convolutional AutoEncoder-based Lossy Image Compression”. In: *2018 Picture Coding Symposium (PCS)*. 2018, pp. 253–257. DOI: 10.1109/PCS.2018.8456308.

[Chu+20] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé, and N. Thuerey. “Learning Temporal Coherence via Self-Supervision for GAN-Based Video Generation”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: 10.1145/3386569.3392457.

[CJ10] M. Chen and H. Jäenicke. “An information-theoretic framework for visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1206–1215.

[CL96] B. Curless and M. Levoy. “A volumetric method for building complex models from range images”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312.

[CM10] C. D. Correa and K.-L. Ma. “Visibility histograms and visibility-driven transfer functions”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.2 (2010), pp. 192–204.

[CS94] D. Cohen and Z. Sheffer. “Proximity clouds – an acceleration technique for 3D grid traversal”. In: *The Visual Computer* 11.1 (1994), pp. 27–38.

[CUH16] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and accurate deep network learning by exponential linear units (elus)”. In: *4th International Conference on Learning Representations (ICLR)*. 2016. DOI: 10.48550/arXiv.1511.07289.

[Cyb89] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[Dad+16] B. Dado, T. R. Kol, P. Bauszat, J.-M. Thiery, and E. Eisemann. “Geometry and attribute compression for voxel scenes”. In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library. 2016, pp. 397–407.

[DC16] S. Di and F. Cappello. “Fast error-bounded lossy HPC data compression with SZ”. In: *2016 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE. 2016, pp. 730–739.

[DCH88] R. A. Drebin, L. Carpenter, and P. Hanrahan. “Volume rendering”. In: *ACM Siggraph Computer Graphics* 22.4 (1988), pp. 65–74.

[Dev+18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint* (2018). DOI: 10.48550/ARXIV.1810.04805.

[Dev85] A. J. Devaney. “Generalized projection-slice theorem for fan beam diffraction tomography”. In: *Ultrasonic Imaging* 7.3 (1985), pp. 264–275. DOI: https://doi.org/10.1016/0161-7346(85)90006-9.

[DH92] J. Danskin and P. Hanrahan. “Fast Algorithms for Volume Ray Tracing”. In: *Proceedings of the 1992 Workshop on Volume Visualization*. VVS ’92. Boston, Massachusetts, USA: ACM, 1992, pp. 91–98. DOI: 10.1145/147130.147155.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7 (2011).

[DMG20] J. Díaz, F. Marton, and E. Gobbetti. “Interactive spatio-temporal exploration of massive time-varying rectilinear scalar volumes based on a variable bit-rate sparse representation over learned dictionaries”. In: *Computers & Graphics* 88 (2020), pp. 45–56.

[DN11] M. Drulea and S. Nedevschi. “Total variation regularization of local-global optical flow”. In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2011, pp. 318–323.

[DNJ20] T. Davies, D. Nowrouzezahrai, and A. Jacobson. “On the effectiveness of weight-encoded neural implicit 3D shapes”. In: *arXiv preprint* (2020). DOI: 10.48550/arXiv.2009.09808.

[Don+16] C. Dong, C. C. Loy, K. He, and X. Tang. “Image super-resolution using deep convolutional networks”. In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2016), pp. 295–307.

[Dos+15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. “FlowNet: Learning optical flow with convolutional networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2758–2766.

[Dre62] S. Dreyfus. “The numerical solution of variational problems”. In: *Journal of Mathematical Analysis and Applications* 5.1 (1962), pp. 30–45. DOI: https://doi.org/10.1016/0022-247X(62)90004-5.

[DV16] V. Dumoulin and F. Visin. “A guide to convolution arithmetic for deep learning”. In: *ArXiv preprint* (2016). DOI: 10.48550/arXiv.1603.07285.

[EHT18] M.-L. Eckert, W. Heidrich, and N. Thuerey. “Coupled Fluid Density and Motion from Single Views”. In: *Computer Graphics Forum* 37.8 (2018), pp. 47–58. DOI: 10.1111/cgf.13511.

[EKE01] K. Engel, M. Kraus, and T. Ertl. “High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading”. In: *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Workshop on Graphics Hardware*. HWWS ’01. Los Angeles, California, USA, 2001, pp. 9–16. DOI: 10.1145/383507.383515.

[ER21] D. Engel and T. Ropinski. “Deep Volumetric Ambient Occlusion”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1268–1278. DOI: 10.1109/TVCG.2020.3030344.

[Eti+13] T. Etiene, D. Jönsson, T. Ropinski, C. Scheidegger, J. L. Comba, L. G. Nonato, R. M. Kirby, A. Ynnerman, and C. T. Silva. "Verifying volume rendering using discretization error analysis". In: *IEEE transactions on visualization and computer graphics* 20.1 (2013), pp. 140–154.

[EUT19] M.-L. Eckert, K. Um, and N. Thuerey. "ScalarFlow: A Large-Scale Volumetric Data Set of Real-World Scalar Transport Flows for Computer Animation and Machine Learning". In: *ACM Transactions on Graphics* 38.6 (2019). DOI: 10.1145/3355089.3356545.

[FH19] M. Feurer and F. Hutter. "Hyperparameter optimization". In: *Automated machine learning*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Springer, 2019, pp. 3–33.

[Fra+19] A. Frasson, M. Ender, S. Weiss, M. Kanzler, A. Pandey, J. Schumacher, and R. Westermann. "Visual Exploration of Circulation Rolls in Convective Heat Flows". In: *2019 IEEE Pacific Visualization Symposium (PacificVis)*. 2019, pp. 202–211. DOI: 10.1109/PacificVis.2019.00031.

[Fri+20] F. Frieß, M. Braun, V. Bruder, S. Frey, G. Reina, and T. Ertl. "Foveated Encoding for Large High-Resolution Displays". In: *IEEE Transactions on Visualization and Computer Graphics* (2020), pp. 1–10. DOI: 10.1109/TVCG.2020.3030445.

[FSK13] T. Fogal, A. Schiewe, and J. Kruger. "An Analysis of Scalable GPU-Based Ray-Guided Volume Rendering". In: *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. Vol. 2013. Oct. 2013, pp. 43–51. DOI: 10.1109/LDAV.2013.6675157.

[Fuk80] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. DOI: 10.1007/BF00344251.

[Gar+21] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. "FastNeRF: High-Fidelity Neural Rendering at 200FPS". In: (2021), pp. 14346–14355.

[GB08] J. Gregor and T. Benson. "Computational Analysis and Improvement of SIRT". In: *IEEE Transactions on Medical Imaging* 27.7 (2008), pp. 918–924. DOI: 10.1109/TMI.2008.923696.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[GBH70] R. Gordon, R. Bender, and G. T. Herman. "Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography". In: *Journal of Theoretical Biology* 29.3 (1970), pp. 471–481. DOI: https://doi.org/10.1016/0022-5193(70)90109-8.

[GEB16] 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 (CVPR)*. 2016, pp. 2414–2423.

[Gei+19] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In: *International Conference on Learning Representations*. 2019.

[GHA05] A. P. Gibson, J. C. Hebden, and S. R. Arridge. “Recent advances in diffuse optical imaging”. In: *Physics in medicine & biology* 50.4 (2005), R1.

[Gho+19] S. Ghosh, A. Pal, S. Jaiswal, K. Santosh, N. Das, and M. Nasipuri. “SegFast-V2: Semantic image segmentation with less parameters in deep learning for autonomous driving”. In: *International Journal of Machine Learning and Cybernetics* 10.11 (2019), pp. 3145–3154.

[Gir15] R. Girshick. “Fast R-CNN”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[Gki+13] I. Gkioulekas, S. Zhao, K. Bala, T. Zickler, and A. Levin. “Inverse Volume Rendering with Material Dictionaries”. In: *ACM Transactions on Graphics* 32.6 (2013). DOI: 10.1145/2508363.2508377.

[GM19] D. Ganter and M. Manzke. “An Analysis of Region Clustered BVH Volume Rendering on GPU”. In: *Computer Graphics Forum*. Vol. 38. 8. Wiley Online Library. 2019, pp. 13–21.

[GMIG08] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. “A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets”. In: *The Visual Computer* 24.7 (2008), pp. 797–806. DOI: 10.1007/s00371-008-0261-9.

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[Gre+12] J. Gregson, M. Krimerman, M. B. Hullin, and W. Heidrich. “Stochastic tomography and its applications in 3D imaging of mixing fluids”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), pp. 1–10.

[Gu+21] P. Gu, J. Han, D. Z. Chen, and C. Wang. “Reconstructing Unsteady Flow Data From Representative Streamlines via Diffusion and Deep-Learning-Based Denoising”. In: *IEEE Computer Graphics and Applications* 41.6 (2021), pp. 111–121. DOI: 10.1109/MCG.2021.3089627.

[Gue+12] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. “Foveated 3D graphics”. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 164.

[Gul+17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. “Improved Training of Wasserstein GANs”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

[Gun+07] J. Gunther, S. Popov, H.-P. Seidel, and P. Slusallek. “Realtime ray tracing on GPU with BVH-based packet traversal”. In: *2007 IEEE Symposium on Interactive Ray Tracing*. IEEE. 2007, pp. 113–118.

[Guo+20] L. Guo, S. Ye, J. Han, H. Zheng, H. Gao, D. Z. Chen, J.-X. Wang, and C. Wang. “SSR-VFD: Spatial Super-Resolution for Vector Field Data Analysis and Visualization”. In: *2020 IEEE Pacific Visualization Symposium (PacificVis)*. 2020, pp. 71–80. DOI: 10.1109/PacificVis48177.2020.8737.

[GW00] A. Griewank and A. Walther. “Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation”. In: *ACM Transactions on Mathematical Software (TOMS)* 26.1 (2000), pp. 19–45.

[GW08] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[Had+05] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross. “Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces”. In: *Comput. Graph. Forum* 24 (2005), pp. 303–312.

[Had+18] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister. “SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 974–983. DOI: 10.1109/TVCG.2017.2744238.

[Hai+10] 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)*. IEEE. 2010, pp. 17–24.

[Han+19] J. Han, J. Tao, H. Zheng, H. Guo, D. Z. Chen, and C. Wang. “Flow field reduction via reconstructing vector data from 3-d streamlines using deep learning”. In: *IEEE computer graphics and applications* 39.4 (2019), pp. 54–67.

[Han+21] J. Han, H. Zheng, Y. Xing, D. Z. Chen, and C. Wang. “V2V: A Deep Learning Approach to Variable-to-Variable Selection and Translation for Multivariate Time-Varying Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1290–1300. DOI: 10.1109/TVCG.2020.3030346.

[Han+22] J. Han, H. Zheng, D. Z. Chen, and C. Wang. "STNet: An End-to-End Generative Framework for Synthesizing Spatiotemporal Super-Resolution Volumes". In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2022), pp. 270–280. DOI: 10.1109/TVCG.2021.3114815.

[Hap+11] M. Hapala, T. Davidovič, I. Wald, V. Havran, and P. Slusallek. "Efficient stack-less BVH traversal for ray tracing". In: *Proceedings of the 27th Spring Conference on Computer Graphics*. 2011, pp. 7–12.

[Har96] J. C. Hart. "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces". In: *The Visual Computer* 12.10 (1996), pp. 527–545.

[Has+20] J. Hasselgren, J. Munkberg, M. Salvi, A. Patney, and A. Lefohn. "Neural temporal adaptive sampling and denoising". In: *Computer Graphics Forum*. Vol. 39. 2. Wiley Online Library. 2020, pp. 147–155.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2016, pp. 770–778.

[He+19] 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". In: *IEEE transactions on visualization and computer graphics* 26.1 (2019), pp. 23–33.

[HED05] T. Hawkins, P. Einarsson, and P. Debevec. "Acquisition of time-varying participating media". In: *ACM Transactions on Graphics (ToG)* 24.3 (2005), pp. 812–815.

[Hed+21] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. "Baking neural radiance fields for real-time view synthesis". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5875–5884.

[Her09] G. T. Herman. *Fundamentals of computerized tomography: image reconstruction from projections*. Springer Science & Business Media, 2009.

[HG41] L. G. Henyey and J. L. Greenstein. "Diffuse radiation in the galaxy". In: *The Astrophysical Journal* 93 (1941), pp. 70–83.

[Hoe16] R. K. Hoetzlein. "GVDB: Raytracing Sparse Voxel Database Structures on the GPU". In: *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by U. Assarsson and W. Hunt. The Eurographics Association, 2016. DOI: 10.2312/hpg.20161197.

[Höh+20] K. Höhlein, M. Kern, T. Hewson, and R. Westermann. “A comparative study of convolutional neural network models for wind field downscaling”. In: *Meteorological Applications* 27.6 (2020), e1961.

[How+17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. “MobileNets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint* (2017). DOI: 10.48550/arXiv.1704.04861.

[HSK89] J. C. Hart, D. J. Sandin, and L. H. Kauffman. “Ray tracing deterministic 3-D fractals”. In: *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. 1989, pp. 289–296.

[HSM00] D. J. Hawrysz and E. M. Sevick-Muraca. “Developments toward diagnostic breast cancer imaging using near-infrared optical measurements and fluorescent contrast agents1”. In: *Neoplasia* 2.5 (2000), pp. 388–417.

[Hsu+19] C.-C. Hsu, C.-W. Lin, W.-T. Su, and G. Cheung. “SiGAN: Siamese generative adversarial network for identity-preserving face hallucination”. In: *IEEE Transactions on Image Processing* 28.12 (2019), pp. 6225–6236. DOI: 10.1109/TIP.2019.2924554.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[HTW20] J. Han, J. Tao, and C. Wang. “FlowNet: A Deep Learning Framework for Clustering and Selection of Streamlines and Stream Surfaces”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.4 (2020), pp. 1732–1744. DOI: 10.1109/TVCG.2018.2880207.

[Hua+17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2017, pp. 4700–4708.

[HW20] J. Han and C. Wang. “TSR-TVD: Temporal Super-Resolution for Time-Varying Data Analysis and Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 205–215. DOI: 10.1109/TVCG.2019.2934255.

[HW22] J. Han and C. Wang. “SSR-TVD: Spatial Super-Resolution for Time-Varying Data Analysis and Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.6 (2022), pp. 2445–2456. DOI: 10.1109/TVCG.2020.3032123.

[Ilg+17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. “FlowNet 2.0: Evolution of optical flow estimation with deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2462–2470.

[Ily+19] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. “Adversarial examples are not bugs, they are features”. In: *Advances in neural information processing systems* 32 (2019).

[IM04] I. Ihrke and M. Magnor. “Image-based tomographic reconstruction of flames”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2004, pp. 365–373.

[Iso+20] T. Isobe, S. Li, X. Jia, S. Yuan, G. Slabaugh, C. Xu, Y.-L. Li, S. Wang, and Q. Tian. “Video super-resolution with temporal group attention”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8008–8017.

[Iza+11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. “KinectFusion: real-time 3d reconstruction and interaction using a moving depth camera”. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 2011, pp. 559–568.

[JAFF16] J. Johnson, A. Alahi, and L. Fei-Fei. “Perceptual losses for real-time style transfer and super-resolution”. In: *European conference on computer vision*. Springer. 2016, pp. 694–711.

[Jar+09] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. “What is the best multi-stage architecture for object recognition?” In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[Jia+21] J. Jiang, C. Wang, X. Liu, and J. Ma. “Deep Learning-Based Face Super-Resolution: A Survey”. In: *ACM Comput. Surv.* 55.1 (2021). DOI: 10.1145/3485132.

[Jo+18] Y. Jo, S. W. Oh, J. Kang, and S. J. Kim. “Deep Video Super-Resolution Network Using Dynamic Upsampling Filters Without Explicit Motion Compensation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 3224–3232.

[Joh+20] J. Johnson, N. Ravi, J. Reizenstein, D. Novotny, S. Tulsiani, C. Lassner, and S. Branson. “Accelerating 3D Deep Learning with PyTorch3D”. In: SA ’20 (2020). DOI: 10.1145/3415263.3419160.

[JS01] M. W. Jones and R. Satherley. “Using distance fields for object representation and rendering”. In: *Proc. 19th Ann. Conf. of Eurographics (UK Chapter)*. London. 2001, pp. 37–44.

[JS06] G. Ji and H.-W. Shen. “Dynamic view selection for time-varying volumes”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 1109–1116.

[JTC14] P. Józsa, M. J. Tóth, and B. Csébfalvi. “Analytic Isosurface Rendering and Maximum Intensity Projection on the GPU”. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Václav Skala-UNION Agency, 2014.

[JZA18] N. Jmour, S. Zayen, and A. Abdelkrim. “Convolutional neural networks for image classification”. In: *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. 2018, pp. 397–402. DOI: 10.1109/ASET.2018.8379889.

[Kal+17] S. Kallweit, T. Müller, B. McWilliams, M. Gross, and J. Novák. “Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 36.6 (Nov. 2017). DOI: 10.1145/3130800.3130880.

[Kap+16] A. Kappeler, S. Yoo, Q. Dai, and A. K. Katsaggelos. “Video super-resolution with convolutional neural networks”. In: *IEEE Transactions on Computational Imaging* 2.2 (2016), pp. 109–122.

[Kap+19] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo. “DeepFovea: neural reconstruction for foveated rendering and video compression using learned statistics of natural videos”. In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–13.

[Kat+20] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. “Differentiable rendering: A survey”. In: *arXiv preprint* (2020). DOI: 10.48550/arXiv.2006.12057.

[KB15] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *3rd International Conference for Learning Representations (ICLR)*. 2015. DOI: 10.48550/arXiv.1412.6980.

[Kim+19] J. Kim, Y. Jeong, M. Stengel, K. Akşit, R. Albert, B. Boudaoud, T. Greer, J. Kim, W. Lopes, Z. Majercik, et al. “Foveated AR: dynamically-foveated augmented reality display”. In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–15.

[KKH01] J. Kniss, G. Kindlmann, and C. Hansen. “Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets”. In: *Proceedings Visualization, 2001. VIS’01.* IEEE. 2001, pp. 255–562.

[KKH05] J. Kniss, G. Kindlmann, and C. D. Hansen. “Functions for Volume Rendering”. In: *Visualization handbook* (2005), p. 189.

[KKLML16] J. Kim, J. Kwon Lee, and K. Mu Lee. “Accurate image super-resolution using very deep convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2016, pp. 1646–1654.

[KKR18] A. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi. “Deep adaptive sampling for low sample count rendering”. In: *Computer Graphics Forum*. Vol. 37. 4. Wiley Online Library. 2018, pp. 35–44.

[Kla+17] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. “Self-normalizing neural networks”. In: *Advances in neural information processing systems* 30 (2017).

[Kle+05] T. Klein, M. Strengert, S. Stegmaier, and T. Ertl. “Exploiting Frame-to-Frame Coherence for Accelerating High-Quality Volume Raycasting on Graphics Hardware”. In: *Proceedings of IEEE Visualization*. IEEE, 2005, pp. 223–230.

[Kni+03] J. Kniss, M. Ikits, A. Lefohn, C. Hansen, E. Praun, et al. “Gaussian transfer functions for multi-field volume visualization”. In: *IEEE Visualization (VIS)*. IEEE. 2003, pp. 497–504.

[Kno+06] A. Knoll, I. Wald, S. Parker, and C. Hansen. “Interactive isosurface ray tracing of large octree volumes”. In: *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE. 2006, pp. 115–124.

[Kno+11] A. Knoll, S. Thelen, I. Wald, C. D. Hansen, H. Hagen, and M. E. Papka. “Full-resolution interactive CPU volume rendering with coherent BVH traversal”. In: *2011 IEEE Pacific Visualization Symposium*. IEEE. 2011, pp. 3–10.

[Kra05] M. Kraus. “Scale-invariant volume rendering”. In: *IEEE Visualization (VIS)*. IEEE. 2005, pp. 295–302.

[Kra09] M. Kraus. “The pull-push algorithm revisited”. In: *Proceedings of the Fourth International Conference on Computer Graphics Theory and Applications (GRAPP)*. 2009, pp. 179–184. DOI: 10.5220/0001772601790184.

[Kra+11] A. Kratz, J. Reininghaus, M. Hadwiger, and I. Hotz. *Adaptive Screen-Space Sampling for Volume Ray-Casting*. eng. Tech. rep. 11-04. URN: urn:nbn:de:0297-zib-12446. Takustr. 7, 14195 Berlin: ZIB, 2011.

[KSA13] V. Kämpe, E. Sintorn, and U. Assarsson. “High resolution sparse voxel dags”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–13.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[KT90] K. Kamijo and T. Tanigawa. “Stock price pattern recognition-a recurrent neural network approach”. In: *1990 IJCNN International Joint Conference on Neural Networks*. Vol. 1. 1990, pp. 215–221. DOI: 10.1109/IJCNN.1990.137572.

[KUH18] H. Kato, Y. Ushiku, and T. Harada. “Neural 3d mesh renderer”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2018, pp. 3907–3916.

[KW03] J. Krüger and R. Westermann. “Acceleration techniques for GPU-based volume rendering”. In: *IEEE Visualization (VIS)*. 2003, pp. 287–292. DOI: 10.1109/VIS.2003.10001.

[KWH09] A. M. Knoll, I. Wald, and C. D. Hansen. “Coherent multiresolution isosurface ray tracing”. In: *The Visual Computer* 25.3 (2009), pp. 209–225.

[Lai+17] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. “Deep laplacian pyramid networks for fast and accurate superresolution”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 2017, p. 5.

[LB14] M. M. Loper and M. J. Black. “OpenDR: An approximate differentiable renderer”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 154–169.

[LC87] W. E. Lorensen and H. E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *ACM siggraph computer graphics* 21.4 (1987), pp. 163–169.

[LC+89] Y. Le Cun, L. Jackel, B. Boser, J. Denker, H. Graf, I. Guyon, D. Henderson, R. Howard, and W. Hubbard. “Handwritten digit recognition: applications of neural network chips and automatic learning”. In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46. DOI: 10.1109/35.41400.

[LC96] C.-C. Lin and Y.-T. Ching. “An efficient volume-rendering algorithm with an analytic approach”. In: *The Visual Computer* 12.10 (1996), pp. 515–526.

[LDC06] P. Longhurst, K. Debattista, and A. Chalmers. “A GPU Based Saliency Map for High-Fidelity Selective Rendering”. In: *Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. AFRIGRAPH ’06. Cape Town, South Africa: ACM, 2006, pp. 21–29. DOI: 10.1145/1108590.1108595.

[LeC+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[Led+17] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. “Photo-realistic single image super-resolution using a generative adversarial network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2017, pp. 4681–4690.

[Lee+09] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 609–616.

[Lev88] M. Levoy. “Display of surfaces from volume data”. In: *IEEE Computer graphics and Applications* 8.3 (1988), pp. 29–37.

[Lev90a] M. Levoy. “Efficient ray tracing of volume data”. In: *ACM Transactions on Graphics (TOG)* 9.3 (1990), pp. 245–261.

[Lev90b] M. Levoy. “Volume rendering by adaptive refinement”. In: *The Visual Computer* 6.1 (1990), pp. 2–7.

[Li+18] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. “Differentiable monte carlo ray tracing through edge sampling”. In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–11.

[Li+19] S. Li, F. He, B. Du, L. Zhang, Y. Xu, and D. Tao. “Fast spatio-temporal residual network for video super-resolution”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10522–10531.

[Lia+15] R. Liao, X. Tao, R. Li, Z. Ma, and J. Jia. “Video super-resolution via deep draft-ensemble learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 531–539.

[Lin+13] S. Lindholm, D. Jönsson, H. Knusson, and A. Ynnerman. “Towards data centric sampling for volume rendering”. In: *Proceedings of SIGRAD 2013; Visual Computing*. 94. 2013, pp. 55–60.

[Liu+17] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, and T. Huang. “Robust video super-resolution with learned temporal dynamics”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2526–2534.

[Liu+19a] H.-T. D. Liu, M. Tao, C.-L. Li, D. Nowrouzezahrai, and A. Jacobson. “Beyond Pixel Norm-Balls: Parametric Adversaries using an Analytically Differentiable Renderer”. In: *International Conference on Learning Representations*. 2019.

[Liu+19b] S. Liu, T. Li, W. Chen, and H. Li. “Soft rasterizer: A differentiable renderer for image-based 3d reasoning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 7708–7717.

[Liu+22] H. Liu, Z. Ruan, P. Zhao, C. Dong, F. Shang, Y. Liu, L. Yang, and R. Timofte. “Video super-resolution based on deep learning: a comprehensive survey”. In: *Artificial Intelligence Review* (2022), pp. 1–55.

[LJ18] A. Lat and C. Jawahar. “Enhancing OCR accuracy with super resolution”. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE. 2018, pp. 3162–3167.

[LJM21] J. Lei, K. Jia, and Y. Ma. “Learning and Meshing from Deep Implicit Surface Networks Using an Efficient Implementation of Analytic Marching”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3135007.

[Lju+16] 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. 3. Wiley Online Library. 2016, pp. 669–691.

[LK10] S. Laine and T. Karras. “Efficient sparse voxel octrees”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2010), pp. 1048–1059.

[LMW21] D. B. Lindell, J. N. Martel, and G. Wetzstein. “AutoInt: Automatic integration for fast neural volume rendering”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 14556–14565.

[Lu+21] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. “Compressive Neural Representations of Volumetric Scalar Fields”. In: *Computer Graphics Forum* (2021).

[M+21] T. Müller, F. Rousselle, J. Novák, and A. Keller. “Real-time Neural Radiance Caching for Path Tracing”. In: *ACM Transactions on Graphics* 40.4 (Aug. 2021), 36:1–36:16. DOI: 10.1145/3450626.3459812.

[M+22] T. Müller, A. Evans, C. Schied, and A. Keller. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Transactions on Graphics* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127.

[MAG19] F. Marton, M. Agus, and E. Gobbetti. “A framework for GPU-accelerated exploration of massive time-varying rectilinear scalar volumes”. In: *Computer Graphics Forum* 38.3 (2019), pp. 53–66. DOI: 10.1111/cgf.13671.

[Mar+04] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. “Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing.” In: *International Symposium on Vision, Modeling, and Visualization (VMV)*. Vol. 4. 2004, pp. 429–435.

[Mar+13] R. Marques, C. Bouville, M. Ribardière, L. P. Santos, and K. Bouatouch. “Spherical Fibonacci Point Sets for Illumination Integrals”. In: *Computer Graphics Forum* 32.8 (2013), pp. 134–143. DOI: https://doi.org/10.1111/cgf.12190.

[Mar+21] J. N. P. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein. “Acorn: adaptive coordinate networks for neural scene representation”. In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–13.

[Max95] N. Max. “Optical models for direct volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 1.2 (1995), pp. 99–108.

[McN+04] A. McNamara, A. Treuille, Z. Popović, and J. Stam. “Fluid Control Using the Adjoint Method”. In: *ACM Transactions on Graphics* 23.3 (Aug. 2004), pp. 449–456. DOI: 10.1145/1015706.1015744.

[Mei+01] M. Meißner, M. Doggett, J. Hirche, and U. Kanus. “Efficient space leaping for ray casting architectures”. In: *Volume Graphics 2001*. Springer. 2001, pp. 149–161.

[Mei+21] D. Meister, S. Ogaki, C. Benthin, M. J. Doyle, M. Guthe, and J. Bittner. “A survey on bounding volume hierarchies for ray tracing”. In: *Computer Graphics Forum*. Vol. 40. 2. Wiley Online Library. 2021, pp. 683–712.

[Mes+19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. “Occupancy networks: Learning 3d reconstruction in function space”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4460–4470.

[Mey+18] S. Meyer, A. Djelouah, B. McWilliams, A. Sorkine-Hornung, M. Gross, and C. Schroers. “Phasenet for video frame interpolation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 498–507.

[MH08] L. Van der Maaten and G. Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[MHN+13] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. Citeseer. 2013, p. 3.

[Mil+20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Computer Vision – ECCV 2020*. 2020, pp. 405–421. DOI: 10.1007/978-3-030-58452-8_24.

[Mit07] M. Mittring. “Finding next gen: Cryengine 2”. In: *ACM SIGGRAPH 2007 courses* (2007), pp. 97–121.

[MKH18] E. Miqueles, N. Koshev, and E. S. Helou. “A Backprojection Slice Theorem for Tomographic Reconstruction”. In: *IEEE Transactions on Image Processing* 27.2 (2018), pp. 894–906. DOI: 10.1109/TIP.2017.2766785.

[ML94] S. R. Marschner and R. J. Lobb. “An evaluation of reconstruction filters for volume rendering”. In: *Proceedings Visualization’94*. IEEE. 1994, pp. 100–107.

[MO74] R. Mersereau and A. Oppenheim. “Digital reconstruction of multidimensional signals from their projections”. In: *Proceedings of the IEEE* 62.10 (1974), pp. 1319–1338. DOI: 10.1109/PROC.1974.9625.

[Mor+19] 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)*. IEEE. 2019, pp. 256–260.

[Müh+07] K. Mühler, M. Neugebauer, C. Tietjen, and B. Preim. “Viewpoint selection for intervention planning.” In: *EuroVis*. 2007, pp. 267–274.

[MYR10] Y. Mukaigawa, Y. Yagi, and R. Raskar. “Analysis of light transport in scattering media”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2010, pp. 153–160.

[Mys98] K. Myszkowski. “The Visible Differences Predictor: applications to global illumination problems”. In: *Rendering Techniques ’98*. Ed. by G. Drettakis and N. Max. Vienna: Springer Vienna, 1998, pp. 223–236.

[NA92] K. Novins and J. Arvo. “Controlled precision volume integration”. In: *Proceedings of the 1992 workshop on Volume visualization*. 1992, pp. 83–89. DOI: 10.1145/147130.147154.

[Nar+06] S. G. Narasimhan, M. Gupta, C. Donner, R. Ramamoorthi, S. K. Nayar, and H. W. Jensen. “Acquiring Scattering Properties of Participating Media by Dilution”. In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH ’06. Boston, Massachusetts: ACM, 2006, pp. 1003–1012. DOI: 10.1145/1179352.1141986.

[NAS92] K. L. Novins, J. Arvo, and D. Salesin. *Adaptive error bracketing for controlled-precision volume rendering*. Tech. rep. Cornell University, 1992.

[Nay+06] S. K. Nayar, G. Krishnan, M. D. Grossberg, and R. Raskar. “Fast Separation of Direct and Global Components of a Scene Using High Frequency Illumination”. In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH ’06. Boston, Massachusetts: ACM, 2006, pp. 935–944. DOI: 10.1145/1179352.1141977.

[ND+19] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. “Mitsuba 2: A Retargetable Forward and Inverse Renderer”. In: *ACM Transactions on Graphics* 38.6 (Nov. 2019). DOI: 10.1145/3355089.3356498.

[ND+20] M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob. “Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020). DOI: 10.1145/3386569.3392406.

[ND+21] M. Nimier-David, Z. Dong, W. Jakob, and A. Kaplanyan. “Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering”. In: *Eurographics Symposium on Rendering - DL-only Track*. The Eurographics Association, 2021. DOI: 10.2312/sr.20211292.

[ND+22] M. Nimier-David, T. Müller, A. Keller, and W. Jakob. “Unbiased Inverse Volume Rendering with Differential Trackers”. In: *ACM Transactions on Graphics* 41.4 (July 2022), 44:1–44:20. DOI: 10.1145/3528223.3530073.

[Nef+21] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. Kaplanyan, and M. Steinberger. “DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks”. In: *Computer Graphics Forum*. Vol. 40. 4. Wiley Online Library. 2021, pp. 45–59.

[Nei10] R. D. Neidinger. “Introduction to automatic differentiation and MATLAB object-oriented programming”. In: *SIAM review* 52.3 (2010), pp. 545–563.

[Neu+02] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. “Cell-Based First-Hit Ray Casting.” In: *VisSym*. 2002, pp. 77–86.

[New+11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. “KinectFusion: Real-time dense surface mapping and tracking”. In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[NH10] V. Nair and G. E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *International Conference on Machine Learning (ICML)*. 2010.

[NH91] G. M. Nielson and B. Hamann. “The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes”. In: *Proceedings of the 2nd Conference on Visualization ’91*. VIS ’91. San Diego, California: IEEE Computer Society Press, 1991, pp. 83–91.

[Nic06] R. W. D. Nickalls. “Viète, Descartes and the cubic equation”. In: *The Mathematical Gazette* 90.518 (2006), pp. 203–208. DOI: 10.1017/S0025557200179598.

[NJJ21] B. Nicolet, A. Jacobson, and W. Jakob. “Large Steps in Inverse Rendering of Geometry”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40.6 (Dec. 2021). DOI: 10.1145/3478513.3480501.

[NL18] S. Niklaus and F. Liu. “Context-aware synthesis for video frame interpolation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 1701–1710.

[NML17a] S. Niklaus, L. Mai, and F. Liu. "Video frame interpolation via adaptive convolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 670–679.

[NML17b] S. Niklaus, L. Mai, and F. Liu. "Video frame interpolation via adaptive separable convolution". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 261–270.

[NP+18] T. H. Nguyen-Phuoc, C. Li, S. Balaban, and Y. Yang. "Rendernet: A deep convolutional network for differentiable rendering from 3d shapes". In: *Advances in Neural Information Processing Systems*. 2018, pp. 7891–7901.

[NVI18] NVIDIA. *NVIDIA Turing GPU Architecture*. `https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf`. Accessed: 2022-05-11. 2018.

[NW99] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[Oka+15] M. Okabe, Y. Dobashi, K. Anjyo, and R. Onai. "Fluid volume modeling from sparse multi-view images by appearance transfer". In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–10.

[Par+18] S.-J. Park, H. Son, S. Cho, K.-S. Hong, and S. Lee. "SRFeat: Single image super-resolution with feature discrimination". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 439–455.

[Par+98] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. "Interactive ray tracing for isosurface rendering". In: *Proceedings Visualization'98 (Cat. No. 98CB36276)*. IEEE. 1998, pp. 233–238.

[Pas+19] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NEURIPS)*. Curran Associates, Inc., 2019, pp. 8024–8035.

[Per+15] S. Perera, N. Barnes, X. He, S. Izadi, P. Kohli, and B. Glocker. "Motion Segmentation of Truncated Signed Distance Function Based Volumetric Surfaces". In: *2015 IEEE Winter Conference on Applications of Computer Vision*. 2015, pp. 1046–1053. DOI: 10.1109/WACV.2015.144.

[Pet+19] F. Petersen, A. H. Bermano, O. Deussen, and D. Cohen-Or. "Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer". In: *arXiv preprint* (2019). DOI: 10.48550/arXiv.1903.11149.

[Por+19] W. P. Porter, Y. Xing, B. R. von Ohlen, J. Han, and C. Wang. “A Deep Learning Approach to Selecting Representative Time Steps for Time-Varying Multivariate Data”. In: *2019 IEEE Visualization Conference (VIS)*. 2019, pp. 1–5. DOI: 10.1109/VISUAL.2019.8933759.

[PS89] 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*. 1989, pp. 281–288.

[Pum+21] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. “D-NeRF: Neural radiance fields for dynamic scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 10318–10327.

[RAHK22] S. R. Richter, H. A. Al Haija, and V. Koltun. “Enhancing photorealism enhancement”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[Rai+20] G. Rainer, A. Ghosh, W. Jakob, and T. Weyrich. “Unified neural encoding of BTFs”. In: *Computer Graphics Forum*. Vol. 39. 2. Wiley Online Library. 2020, pp. 167–178.

[Rai+22] G. Rainer, A. Bousseau, T. Ritschel, and G. Drettakis. “Neural Precomputed Radiance Transfer”. In: *Computer Graphics Forum*. 2022.

[Rei+12] F. Reichl, M. G. Chajdas, K. Bürger, and R. Westermann. “Hybrid Sample-based Surface Rendering”. In: *Vision, Modeling and Visualization*. The Eurographics Association, 2012. DOI: 10.2312/PE/VMV/VMV12/047-054.

[Rei+19] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, et al. “Deep learning and process understanding for data-driven Earth system science”. In: *Nature* 566.7743 (2019), pp. 195–204.

[Rey+85] R. Reynolds, G. T.Herman, J. Udupa, and L. Che. “Surface Shading in the Cuberille Environment”. In: *IEEE Computer Graphics and Applications* 5.12 (1985), pp. 33–43. DOI: 10.1109/MCG.1985.276275.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[RFS03] J. Rigau, M. Feixas, and M. Sbert. “Refinement criteria based on f-divergences”. In: *Rendering Techniques*. 2003, pp. 260–269.

[Rho+15] 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*. 2015, pp. 765–773.

[RHW85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[RKE00] S. Rottger, M. Kraus, and T. Ertl. "Hardware-accelerated volume and isosurface rendering based on cell-projection". In: *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*. 2000, pp. 109–116. DOI: 10.1109/VISUAL.2000.885683.

[Rob20] M. Roberts. *The Unreasonable Effectiveness of Quasirandom Sequences*. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/. Accessed: 2022-05-14. 2020.

[RPG99] M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. "A Perceptually Based Physical Error Metric for Realistic Image Synthesis". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. ACM, 1999, pp. 73–82. DOI: 10.1145/311535.311543.

[Rui+11] M. Ruiz, A. Bardera, I. Boada, and I. Viola. "Automatic transfer functions based on informational divergence". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 1932–1941.

[Rus+15] O. Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[SA07] P. Shanmugam and O. Arikan. "Hardware accelerated ambient occlusion techniques on GPUs". In: *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 2007, pp. 73–80.

[SB21] S. Sahoo and M. Berger. "Integration-Aware Vector Field Super Resolution". In: *EuroVis 2021 - Short Papers*. Ed. by M. Agus, C. Garth, and A. Kerren. The Eurographics Association, 2021. DOI: 10.2312/evs.20211054.

[Sch13] J. Schwarze. "Cubic and Quartic Roots". In: *Graphics gems*. Ed. by A. S. Glassner. Elsevier, 2013. Chap. VIII.1, pp. 404–407.

[Shi+16] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1874–1883.

[Sil+17] I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves. “Forecast of Stock Market Trends Using Recurrent Networks”. In: *Artificial Neural Networks : A Practical Course*. Cham: Springer International Publishing, 2017, pp. 221–227. DOI: 10.1007/978-3-319-43162-8_13.

[SK94] L. M. Sobierajski and A. E. Kaufman. “Volumetric Ray Tracing”. In: *Proceedings of the 1994 Symposium on Volume Visualization*. VVS ’94. Tysons Corner, Virginia, USA: ACM, 1994, pp. 11–18. DOI: 10.1145/197938.197949.

[SMK05] S. M. Seitz, Y. Matsushita, and K. N. Kutulakos. “A theory of inverse light transport”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Vol. 2. IEEE. 2005, pp. 1440–1447.

[Sri+21] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron. “NeRV: Neural reflectance and visibility fields for relighting and view synthesis”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 7495–7504.

[SSH17] M. S. Sajjadi, B. Scholkopf, and M. Hirsch. “EnhanceNet: Single image super-resolution through automated texture synthesis”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 4491–4500.

[ST19] N. Shi and Y. Tao. “CNNs based viewpoint estimation for volume visualization”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.3 (2019), pp. 1–22.

[Sut+13] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

[SVB18] M. S. Sajjadi, R. Vemulapalli, and M. Brown. “Frame-recurrent video super-resolution”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6626–6634.

[SW88] H. Samet and R. E. Webber. “Hierarchical data structures and algorithms for computer graphics. I. Fundamentals”. In: *IEEE Computer Graphics and applications* 8.3 (1988), pp. 48–68.

[Sze+14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. “Intriguing properties of neural networks”. In: *International Conference on Learning Representations (ICLR)*. 2014.

[Szt+21] A. Sztrajman, G. Rainer, T. Ritschel, and T. Weyrich. “Neural BRDF Representation and Importance Sampling”. In: *Computer Graphics Forum*. Vol. 40. 6. Wiley Online Library. 2021, pp. 332–346.

[SZW19] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. “Scene representation networks: Continuous 3D-structure-aware neural scene representations”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 1119–1130.

[Tak+05] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. “A feature-driven approach to locating optimal viewpoints for volume visualization”. In: *IEEE Visualization (VIS)*. IEEE. 2005, pp. 495–502.

[Tak+21] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. “Neural geometric level of detail: Real-time rendering with implicit 3D shapes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11358–11367.

[Tan+20] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7537–7547.

[Tao+09] Y. Tao, H. Lin, H. Bao, F. Dong, and G. Clapworthy. “Structure-aware viewpoint selection for volume visualization”. In: *2009 IEEE Pacific Visualization Symposium*. IEEE. 2009, pp. 193–200.

[Tao+16] Y. Tao, Q. Wang, W. Chen, Y. Wu, and H. Lin. “Similarity voting based viewpoint selection for volumes”. In: *Computer graphics forum*. Vol. 35. 3. Wiley Online Library. 2016, pp. 391–400.

[Tao+17] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia. “Detail-Revealing Deep Video Super-Resolution”. In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[Tew+20] 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. 2. Wiley Online Library. 2020, pp. 701–727.

[Tew+22] A. Tewari et al. “Advances in Neural Rendering”. In: *Computer Graphics Forum* (2022). DOI: 10.1111/cgf.14507.

[TFE19] G. Tkachev, S. Frey, and T. Ertl. “Local prediction models for spatiotemporal volume visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.7 (2019), pp. 3091–3108.

[The+17] L. Theis, W. Shi, A. Cunningham, and F. Huszár. “Lossy Image Compression with Compressive Autoencoders”. In: *5th International Conference on Learning Representations (ICLR)*. 2017. DOI: 10.48550/ARXIV.1703.00395.

[Tie+90] U. Tiede, K. Hoehne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. “Investigation of medical 3D-rendering algorithms”. In: *IEEE Computer Graphics and Applications* 10.2 (1990), pp. 41–53. DOI: 10.1109/38.50672.

[Ton+17] T. Tong, G. Li, X. Liu, and Q. Gao. “Image super-resolution using dense skip connections”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4799–4807.

[TPG99] G. M. Treece, R. W. Prager, and A. H. Gee. “Regularised marching tetrahedra: improved iso-surface extraction”. In: *Computers & Graphics* 23.4 (1999), pp. 583–598.

[Tre+12] M. Treib, K. Bürger, F. Reichl, C. Meneveau, A. Szalay, and R. Westermann. “Turbulence Visualization at the Terascale on Desktop PCs”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2169–2177. DOI: 10.1109/TVCG.2012.274.

[Tur+19] O. T. Tursun, E. Arabadzhiyska-Koleva, M. Wernikowski, R. Mantiuk, H.-P. Seidel, K. Myszkowski, and P. Didyk. “Luminance-Contrast-Aware Foveated Rendering”. In: *ACM Transactions on Graphics* 38.4 (July 2019). DOI: 10.1145/3306346.3322985.

[TZN19] J. Thies, M. Zollhöfer, and M. Nießner. “Deferred neural rendering: Image synthesis using neural textures”. In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[UH83] J. Udupa and G. Herman. “Display of 3-D Digital Images: Computational Foundations and Medical Applications”. In: *IEEE Computer Graphics and Applications* 3.05 (1983), pp. 39–46. DOI: 10.1109/MCG.1983.263213.

[van+15] 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”. In: *Ultramicroscopy* 157 (2015), pp. 35–47. DOI: https://doi.org/10.1016/j.ultramic.2015.05.002.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[Váz+01] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. “Viewpoint selection using viewpoint entropy.” In: *VMV*. Vol. 1. Citeseer. 2001, pp. 273–280.

[VMN08] P.-P. Vázquez, E. Monclús, and I. Navazo. “Representative views and paths for volume models”. In: *International Symposium on Smart Graphics*. Springer. 2008, pp. 106–117.

[VSJ21] D. Vicini, S. Speierer, and W. Jakob. “Path replay backpropagation: differentiating light paths using constant memory and linear time”. In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–14.

[WAH93] J. Weng, N. Ahuja, and T. Huang. “Learning recognition and segmentation of 3-D objects from 2-D images”. In: *1993 (4th) International Conference on Computer Vision*. 1993, pp. 121–128. DOI: 10.1109/ICCV.1993.378228.

[Wal00] J. Walden. “Analysis of the direct Fourier method for computer tomography”. In: *IEEE transactions on Medical Imaging* 19.3 (2000), pp. 211–222.

[Wan+04] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[Wan+18] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy. “ESRGAN: Enhanced super-resolution generative adversarial networks”. In: *Proceedings of the European conference on computer vision (ECCV) workshops*. 2018, pp. 0–0.

[WCH21] Z. Wang, J. Chen, and S. C. H. Hoi. “Deep Learning for Image Super-Resolution: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2021), pp. 3365–3387. DOI: 10.1109/TPAMI.2020.2982166.

[Wei+20] S. Weiss, M. Işık, J. Thies, and R. Westermann. “Learning Adaptive Sampling and Reconstruction for Volume Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* (2020), pp. 1–1. DOI: 10.1109/TVCG.2020.3039340.

[Wei+21] S. Weiss, M. Chu, N. Thuerey, and R. Westermann. “Volumetric Isosurface Rendering with Deep Learning-Based Super-Resolution”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.6 (2021), pp. 3064–3078. DOI: 10.1109/TVCG.2019.2956697.

[Wen64] R. E. Wengert. “A simple automatic derivative evaluation program”. In: *Communications of the ACM* 7.8 (1964), pp. 463–464.

[WH22] C. Wang and J. Han. “DL4SciVis: A State-of-the-Art Survey on Deep Learning for Scientific Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* (2022), pp. 1–1. DOI: 10.1109/TVCG.2022.3167896.

[WHW21] S. Weiss, P. Hermüller, and R. Westermann. “Fast Neural Representations for Direct Volume Rendering”. In: *arXiv preprint* (2021). DOI: 10.48550/arXiv.2112.01579.

[WK18] E. Wong and Z. Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5286–5295.

[WKB99] M. Wan, A. Kaufman, and S. Bryson. “High performance presence-accelerated ray casting”. In: *Proceedings of the conference on Visualization ’99*. 1999, pp. 379–386.

[WM92] P. L. Williams and N. Max. “A volume density optical model”. In: *Proceedings of the 1992 workshop on Volume visualization*. 1992, pp. 61–68.

[WSK18] C.-Y. Wu, N. Singhal, and P. Krahenbuhl. “Video compression through image interpolation”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 416–431.

[Wur+21] S. W. Wurster, H.-W. Shen, H. Guo, T. Peterka, M. Raj, and J. Xu. “Deep Hierarchical Super-Resolution for Scientific Data Reduction and Visualization”. In: *arXiv preprint* (2021). DOI: 10.48550/arXiv.2107.00462.

[WW21] S. Weiss and R. Westermann. “Analytic Ray Splitting for Controlled Precision DVR”. In: *EuroVis 2021 - Short Papers*. Ed. by M. Agus, C. Garth, and A. Kerren. The Eurographics Association, 2021. DOI: 10.2312/evs.20211051.

[WW22] S. Weiss and R. Westermann. “Differentiable Direct Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2022), pp. 562–572. DOI: 10.1109/TVCG.2021.3114769.

[WZM21] I. Wald, S. Zellmann, and N. Morrical. “Faster RTX-Accelerated Empty Space Skipping using Triangulated Active Region Boundary Geometry”. In: *Eurographics Symposium on Parallel Graphics and Visualization*. 2021.

[Xie+18] Y. Xie, E. Franz, M. Chu, and N. Thuerey. “tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow”. In: *ACM Transactions on Graphics* 37.4 (2018).

[Xu+05] 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*. Springer. 2005, pp. 989–994.

[Xu+15] B. Xu, N. Wang, T. Chen, and M. Li. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint* (2015). DOI: 10.48550/arXiv.1505.00853.

[XYL20] M.-C. Xu, F. Yin, and C.-L. Liu. “SRR-GAN: Super-Resolution based Recognition with GAN for Low-Resolved Text Images”. In: *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2020, pp. 1–6. DOI: 10.1109/ICFHR2020.2020.00012.

[Yan+19a] C. Yang, Y. Li, C. Liu, and X. Yuan. “Deep learning-based viewpoint recommendation in volume visualization”. In: *Journal of Visualization* 22.5 (2019), pp. 991–1003.

[Yan+19b] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao. “Deep learning for single image super-resolution: A brief review”. In: *IEEE Transactions on Multimedia* 21.12 (2019), pp. 3106–3121.

[You+19] C. You, G. Li, Y. Zhang, X. Zhang, H. Shan, M. Li, S. Ju, Z. Zhao, Z. Zhang, W. Cong, et al. “CT super-resolution GAN constrained by the identical, residual, and cycle learning ensemble (GAN-CIRCLE)”. In: *IEEE transactions on medical imaging* 39.1 (2019), pp. 188–203.

[YR19] Y. Yang and M. Rinard. “Correctness verification of neural networks”. In: *arXiv preprint* (2019). DOI: 10.48550/arXiv.1906.01030.

[Yu+21] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. “Plenoxels: Radiance Fields without Neural Networks”. In: *arXiv preprint arXiv:2112.05131* (2021).

[ZAM11] Z. Zheng, N. Ahmed, and K. Mueller. “iView: A feature clustering framework for suggesting informative views in volume visualization”. In: *IEEE transactions on visualization and computer graphics* 17.12 (2011), pp. 1959–1968.

[Zei12] M. D. Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint* (2012). DOI: 10.48550/arXiv.1212.5701.

[Zha+18] 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.

[Zha+20] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen, and F. Cappello. “Significantly Improving Lossy Compression for HPC Datasets with Second-Order Prediction and Parameter Optimization”. In: *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC ’20. Stockholm, Sweden: ACM, 2020, pp. 89–100.

[Zho+17] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. “Volume Upscaling with Convolutional Neural Networks”. In: *Proceedings of the Computer Graphics International Conference*. CGI ’17. Yokohama, Japan: ACM, 2017, 38:1–38:6. DOI: 10.1145/3095140.3095178.

[ZW10] Y. Zhang and B. Wang. “Optimal viewpoint selection for volume rendering based on shuffled frog leaping algorithm”. In: *2010 IEEE International Conference on Progress in Informatics and Computing*. Vol. 2. IEEE. 2010, pp. 706–709.

Volumetric Isosurface Rendering with Deep Learning-Based Super-Resolution

Sebastian Weiss*, Mengyu Chu†, Nils Thuerey‡ and Rüdiger Westermann§
Technical University of Munich.
Email: *sebastian13.weiss@tum.de, †mengyu.chu@tum.de, ‡nils.thuerey@tum.de, §westermann@tum.de

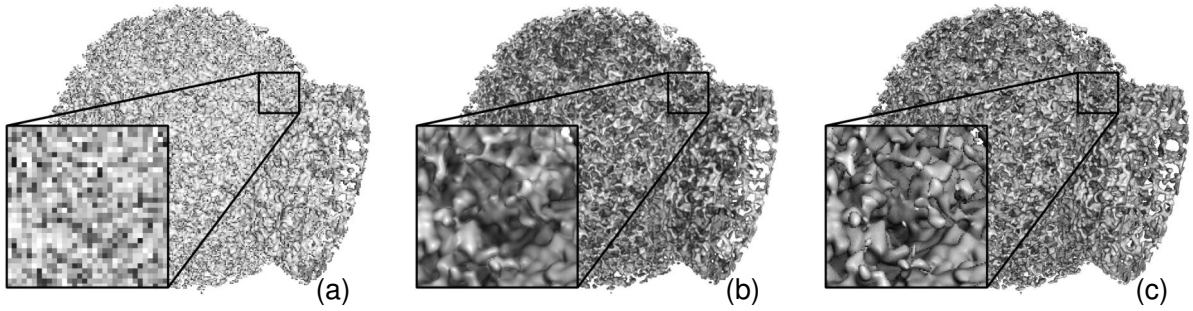


Fig. 1: Our super-resolution network can upscale (a) an input sampling of isosurface depths and normals at low resolution (i.e., 320x240), to (b) a high resolution depth and normal map (i.e., 1280x960) with ambient occlusion. For ease of interpretation, only the shaded output is shown. (c) The ground truth is rendered at 1280x960. Samples are from a 1024^3 grid, ground truth renders at 0.16 and 18.6 secs w/ and w/o ambient occlusion, super-resolution takes 0.07 sec.

Abstract—Rendering an accurate image of an isosurface in a volumetric field typically requires large numbers of data samples. Reducing this number lies at the core of research in volume rendering. With the advent of deep learning networks, a number of architectures have been proposed recently to infer missing samples in multi-dimensional fields, for applications such as image super-resolution. In this paper, we investigate the use of such architectures for learning the upscaling of a low resolution sampling of an isosurface to a higher resolution, with reconstruction of spatial detail and shading. We introduce a fully convolutional neural network, to learn a latent representation generating smooth, edge-aware depth and normal fields as well as ambient occlusions from a low resolution depth and normal field. By adding a frame-to-frame motion loss into the learning stage, upscaling can consider temporal variations and achieves improved frame-to-frame coherence. We assess the quality of inferred results and compare it to bi-linear and -cubic upscaling. We do this for isosurfaces which were never seen during training, and investigate the improvements when the network can train on the same or similar isosurfaces. We discuss remote visualization and foveated rendering as potential applications.

Index Terms—Machine Learning ; Extraction of Surfaces (Isosurfaces, Material Boundaries) ; Volume Rendering.

1 Introduction

Much of the research in isosurface volume ray-casting has been devoted to the development of efficient search structures, i.e., data structures that can reduce the number of data samples required to determine where a view ray intersects the surface. Despite their high degree of sophistication, for large volumes with heterogeneous composition the traversal of these data structures becomes increasingly costly. Since the workload of ray-casting is linear in the number of pixels, frame rates can drop significantly when isosurfaces in large volumes are rendered on high resolution display systems.

This effect is intensified if global illumination effects are considered. An important global illumination effects for isosurfaces is ambient occlusion (AO). AO estimates for every surface point the attenuation of ambient light from the surrounding, and uses this information to enhance cavities and locations closely surrounded by other surface parts. AO is simulated by testing along many secondary rays per surface point whether the isosurface is hit, requiring so many data samples, in general, that interactive frame rates cannot be maintained.

In this work, we investigate the potential of convolutional neural networks to further reduce the number of samples in isosurface ray-casting, for both the reconstruction of the surfaces' geometry and ambient occlusions on it. This strategy works in tandem with an acceleration structure to even more aggressively reduce the number of samples. First, we shed light on the question whether an accurate high resolution image of the surface—a super-resolution image—can be inferred from only the surface points at a far lower image resolution. Second, we aim at investigating whether ambient occlusions can be inferred from the surface geometry without the need to explicitly compute occlusions on that geometry.

From a signal theoretical point of view, it can be argued that new structures—beyond what can be predicted from multiple frames of low resolution inputs by classical up-scaling filters like bi-linear or -cubic interpolation—cannot be inferred without any further assumptions about their occurrence. Recent works in deep learning have demonstrated that such assumptions can be learned by an artificial neural network. Learning-based image and video super-resolution have achieved remarkable results, by training networks

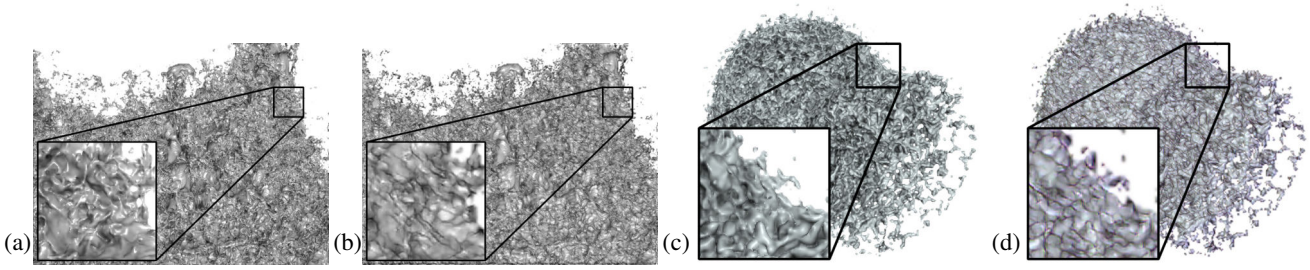


Fig. 2: Super-resolution on depth and normal maps with screen-space shading (a,c) leads to superior reconstruction quality compared to super-resolution on color images (b,d). These images exhibit color bleeding and shifts, while our screen-space shading approach successfully prevents these artifacts. (b) was converted to gray-scale to emphasize geometric differences.

using corresponding pairs of low- and high resolution color images [1], [2]. Learned assumptions can then be transferred to a new low resolution input, to generate a high resolution variant that adheres to the structures seen at training time.

Similar observations have been reported in current works in visualization, which demonstrate the use of neural-network-based inference of data samples for volume upscaling and in-situ visualization, as well as parameter-space exploration. Networks are used to infer high resolution scalar fields and missing time-steps between 3D simulation results [3], [4], and they learn the dependencies between simulation results and the simulation parameters to infer the results for new parameter settings [5]. These works demonstrate that a network can learn to infer new data from given samples, by learning either some type of interpolation, or the local effects of parameter value modifications constraint by the global data distribution.

1.1 Contribution

We present an artificial neural network that learns to upscale a sampled representation of geometric properties of an isosurface at low resolution to a higher resolution. We introduce a fully convolutional neural network, using the FRVSR-Net [6] as a basis, to learn a latent representation that generates a smooth, edge-aware depth and normal field, as well as ambient occlusions, from a low resolution depth and normal field. To support user navigation, we integrate a loss function into the training pass that penalizes frame-to-frame inconsistencies and achieves improved temporal coherence in the reconstruction step.

Even though similar in spirit to classical super-resolution techniques, we strive for a conceptually different approach in this work: Instead of using color images and down-scaled ground truth images for training, we aim at incorporating 3D scene information in the form of per-frame depth and normal fields into the training and reconstruction process. It is our goal to let the network learn the relations between the isosurface geometry sampled at a low and a high resolution, and to infer on the relations between the geometry and shading.

We use the neural network to infer images of isosurfaces with four times the resolution of the input images. Figure 1 demonstrates the result of the upscaling process. Since the network is designed to learn high resolution ambient occlusions from low resolution depth and normal images, computations of ambient occlusions at runtime are entirely avoided. Thus, compared to volumetric ray-casting at full resolution, the number of samples from the volumetric field can be reduced drastically.

To analyse the pixel-wise reconstruction error of the network, we compare reconstructed images to ground truth renderings and reconstructions using bi-linear and bi-cubic upscaling. These comparison are performed using the peak signal-to-noise ratio (PSNR) between ground truth and reconstructed results, as well as the structure-similarity metric (SSIM) [7] that gives more weight to the perceived quality of the results. We demonstrate very good reconstruction quality even for isosurfaces that were never shown to the network during training, and that the network's accuracy can be even improved by retraining on isosurfaces of shapes similar to the ones used in the inference step.

Our specific contributions are:

- We show that it is beneficial to train the network based on depth and normal images instead of color. Our results indicate that this training process results in an improved learning of geometric surface properties, as illustrated in Figure 2.
- Instead of letting the network learn to infer AO in the high resolution output from AO in the low resolution inputs, our networks only receive low resolution depth and normal maps as input. Thus, AO does not need to be simulated at the samples of the low resolution input, which would significantly increase the rendering time.
- To let the network learn to maintain frame-to-frame coherence, we additionally add a motion loss for the generated image content. In this way, the network achieves improved reconstruction quality and becomes well suited for interactive exploration tasks.
- We perform a quality evaluation to shed light on the reconstruction accuracy of learning-based isosurface reconstruction. This evaluation shows the strengths and weaknesses of this type of reconstruction, and helps to better understand its specific properties and possible application scenarios.

For a number of isosurfaces with vastly different geometric properties, we demonstrate the potential of learning-based upscaling. Furthermore, we discuss several use cases where isosurface inference can significantly improve and accelerate existing approaches, e.g., during interactive navigation in remote visualization environments, in focus+context visualization, and in foveated rendering.

2 Related Work

Our approach works in combination with established acceleration techniques for volumetric ray-casting of isosurfaces, and builds upon recent developments in image and video super-resolution

via artificial neural networks to further reduce the number of data access operations.

Volumetric Ray-Casting of Isosurfaces: Over the last decades, considerable effort has been put into the development of acceleration techniques for isosurface ray-casting in 3D scalar fields. Direct volume ray-casting of isosurfaces was proposed by Levoy [8]. Classical fixed-step ray-casting traverses the volume along a ray using equidistant steps in the order of the voxel size. Acceleration structures for isosurface ray-casting encode larger areas were the surface cannot occur, and ray-casting uses this information to skip these areas with few steps. One of the most often used acceleration structure is the min-max pyramid [9], a tree data structure that stores at every interior node the interval of data values in the corresponding part of the volume. Pyramidal data structures are at the core of most volumetric ray-casting techniques to effectively reduce the number of data samples that need to be accessed during ray traversal.

For selected isosurfaces, bounding cells or simple geometries were introduced to restrict ray-traversal to a surface's interior [10], [11]. Adaptive step-size control according to pre-computed distance information aimed at accelerating first-hit determination [12]. Recently, SparseLeap [13] introduced pyramidal occupancy histograms to generate geometric structures representing non-empty regions. They are then rasterized into per-pixel fragment lists to obtain those segments that need to be traversed.

Significant performance improvements have been achieved by approaches which exploit high memory bandwidth and texture mapping hardware on GPUs for sampling and interpolation in 3D scalar fields [14], [15]. For isosurface ray-casting, frame to frame depth buffer coherence on the GPU was employed to speed up first-hit determination [16], [17]. A number of approaches have shown the efficiency of GPU volume ray-casting when paired with compact isosurface representations, brick-based or octree subdivision, and out-of-core strategies for handling data sets too large to be stored on the GPU [18], [19], [20], [21]. For a thorough overview of GPU approaches for large-scale volume rendering, let us refer to the report by Beyer *et al.* [22].

Related to isosurface rendering is the simulation of realistic surface shading effects. AO estimates for every surface point the integral of the visibility function over the hemisphere [23]. AO can greatly improve isosurface visualization, by enhancing the perception of small surface details. A number of approximations for AO simulation in volumetric data sets have been proposed, for instance, local and moment-based approximations of occluding voxels [24], [25] or pre-computed visibility information [26]. The survey by Ropinski *et al.* [27] provides a thorough overview of the use of global illumination in volume visualization. Even though very efficient screen-space approximations of AO exist [28], [29], we decided to consider ray-traced AO in object-space to achieve high quality.

Deep Learning of Super-Resolution and Shading: For super-resolution of natural images, deep learning based methods have progressed rapidly since the very first method [1] surpassed traditional techniques in terms of peak signal-to-noise ratio (PSNR). Regarding network architectures, Kim et al. introduced a very deep network [30], Lai et al. designed the Laplacian pyramid network [31], and advanced network structures have been applied, such as the ResNet [32], [33] and DenseNet [34], [35] architectures. Regarding loss formulations, realistic high-frequency detail is significantly improved by using adversarial and perceptual losses based on pretrained networks [33], [36]. Compared to single-image methods, video super-resolution tasks introduce the time dimensions, and as such require temporal coherence and consistent image content across multiple frames. While many methods use multiple low resolution frames [37], [38], [39], the FRVSR-Net [6] reuses the previously generated high resolution image to achieve better temporal coherence. By using a spatio-temporal discriminator, the TecoGAN [40] network produced results with spatial detail without sacrificing temporal coherence. Overall, motion compensation represents a critical component when taking into account multiple input frames. Methods either use explicit motion estimation and rely on its accuracy [6], [40], [41], [42], or spend extra efforts implicitly such as detail fusion [37] and dynamic upsampling [39]. In our setting, we can instead leverage the computation of reliable screen-space motions via raytracing.

In a different scenario, neural networks were trained to infer images from a noisy input generated via path-tracing with low number of paths, of the same resolution as the target, but with significantly reduced variance in the color samples [43], [44]. Deep shading [45] utilized a neural network to infer shading from rendered images, targeting attributes like position, normals, and reflections for color images of the same resolution. None of these techniques used neural networks for upscaling as we do, yet they are related in that they use additional parameter buffers to improve reconstruction quality of global illumination.

Deep Learning of Volumetric Fields: For volume visualization, Zhou et al. [3] presented a learning-based approach for volume upscaling which better preserves structural details and volume quality than linear upscaling. Berger et al. [46] proposed a deep image synthesis approach to assist transfer function design using generative adversarial networks (GANs). Recently, the use of convolutional neural networks for temporal upscaling has been introduced [4]. The authors demonstrate the application of learning-based reconstruction for in-situ visualization, by letting a network learn to infer the time evolution of a physical field in-between a pair of simulated time steps. In another work it has been demonstrated that a neural network can learn the relationships between simulation parameters and the simulation results [5]. By using training samples consisting of parameter sets and corresponding results, the network can infer the parameter dependencies and use the build latent representation to generate results for new input values.

3 Isosurface Learning

Our method consists of a pre-process in which an artifical neural network is trained, and the upscaling process which receives a new low resolution isosurface image and uses the trained network to perform the upscaling of this image. Our network is designed to perform $4\times$ upscaling, i.e. from input images of size $H \times W$ to output images of size $4H \times 4W$. Note, however, that other upscaling factors can be realized by straight forward adaptations and network re-training.

The network is trained on unshaded surface points. It receives the low resolution input image in the form of a depth and normal map for a selected view, as well as corresponding high resolution maps with an additional AO map that is generated for that view. A low- and high resolution binary mask indicate those pixels where the surface is hit. Once a new low resolution input image is upscaled, i.e., high resolution depth, normal and AO maps are reconstructed, screen-space shading is computed and added to AO in a post-process to generate the final color.

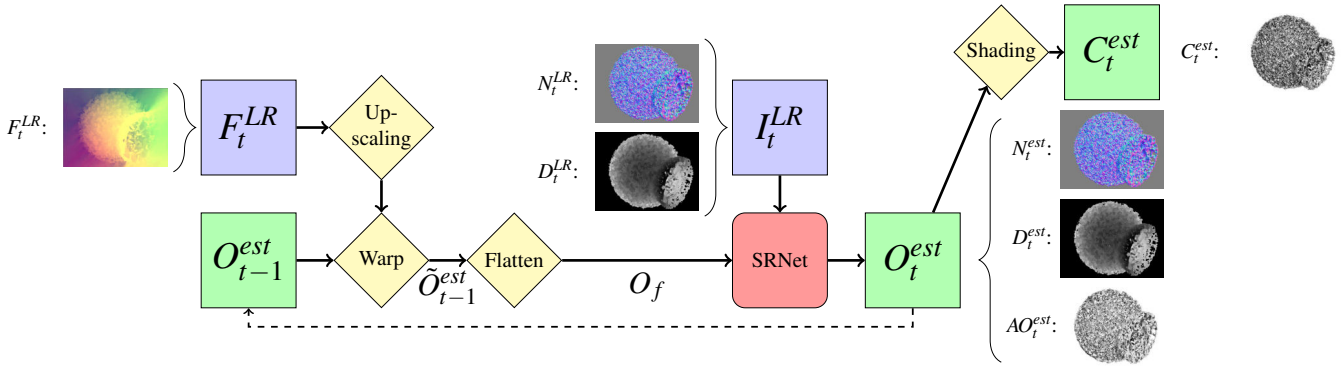


Fig. 3: Overview of network-based learning of isosurface image upscaling. Blue: low resolution inputs, green: high resolution outputs, yellow: fixed processing steps, red: trained network. From left to right: The current optical flow F_t^{LR} is used to warp the output of the previous time step O_{t-1}^{est}. The current low resolution input I_t^{LR} and the warped previous output is given as input to the network. The network produces the output O_t^{est} including estimated high resolution mask (M_t^{est}, not shown in the figure), normal (N_t^{est}), depth (D_t^{est}), and AO (AO_t^{est}) maps . Deferred shading is used to generate the final color (C_t^{est}).

The network, given many low- and high resolution pairs of input maps for different isosurfaces and views, internally builds a so-called latent representation that aims at mapping the low resolution inputs to their high resolution counterparts. A loss function is used to penalize differences between the high resolution learned and ground-truth variants. We investigate different networks, trained with collections of randomly sampled views from a small number of exemplary datasets. Images of isosurfaces at full resolution are used as ground-truth training data. We analyze the upscaling quality of these networks on new views for the training data, as well as views of isosurfaces in datasets the networks have never seen during training.

3.1 Input Data

Both the low- and high resolution input maps are generated via volumetric ray-casting. AO in the high resolution image is simulated by spawning 512 additional secondary rays per surface point, and testing each of them for an intersection with the surface. Since we aim at supporting temporally coherent super-resolution, all images have a time subscript t, starting with $t = 1$ at the first frame.

The following low resolution input maps of size $H \times W$ are used in the training step:

- $M_t^{LR} \in [-1,+1]^{H\times W}$: The binary input mask that specifies for every pixel whether the isosurface is hit (mask=1) or not (mask=-1). Internally, the network learns continuous values, and uses these values to smoothly blend the final color over the background.
- $N_t^{LR} \in [-1,+1]^{3\times H\times W}$: The normal map with the normal vectors in screen-space.
- $D_t^{LR} \in [0,1]^{H\times W}$: The depth map, in which 0 indicates that no hit was determined.

The low resolution input to the network is then given by $I_t^{LR} := \{M_t^{LR}, N_t^{LR}, D_t^{LR}\} \in \mathbb{R}^{5\times H\times W}$. We subsequently call this the low resolution input image.

In addition, the following map is generated during ray-casting:

- $F_t^{LR} \in [-1,+1]^{2\times H\times W}$: A map of 2D displacement vectors, indicating the screen-space flow from the previous view to the current view.

The screen-space flow is used to align the previous high resolution maps with the current low resolution input maps. Under the assumption of temporal coherence, the network can then minimize for the deviation of the currently inferred high resolution map from the temporally extrapolated previous one in the training process. To compute the screen-space flow, assume that in the low resolution view the current ray hits the isosurface at world position x_t. Since during rendering the current and previous model-view-projection matrices are known, the current and previous screen-space coordinates of the point x_t, i.e., x_t' and x_{t-1}', can be computed. The flow is then computed as $f_t := x_t' - x_{t-1}'$, indicating how to displace the previous mask, depth and normal maps at time $t-1$ to align them with the frame at time t. Since the described method provides the displacement vectors only at locations in the low resolution input image where the isosurface is hit in the current frame, we use a Navier-Stokes-based image inpainting [47] via OpenCV [48] to obtain a dense displacement field F_t^{LR}. The inpainting algorithm fills the empty regions in such a way that the resulting flow is as incompressible as possible.

For aligning the previous maps, the current flow field is first upscaled via bi-linear interpolation to the high resolution. In a semi-Lagrangian fashion, we generate new high resolution maps where every pixel in the upscaled maps retrieves the value in the corresponding high resolution map from the previous frame, by using the inverse flow vector to determine the target location.

The high resolution input data, which is used as ground truth in the training process, is comprised of the same maps as the low resolution input, plus an AO map $AO_t^{GT} \in [0,1]^{4H\times 4W}$. Here, values of one or zero indicate no or full occlusion, respectively. Thus, the ground truth image can be written as $O_t^{GT} := \{M_t^{GT}, N_t^{GT}, D_t^{GT}, AO_t^{GT}\} \in \mathbb{R}^{6\times 4H\times 4W}$. Once the network is trained with I_t^{LR} and O_t^{GT}, it can infer a new high resolution output image $O_t^{est} := \{M_t^{est}, N_t^{est}, D_t^{est}, AO_t^{est}\} \in \mathbb{R}^{6\times 4H\times 4W}$ from a given low resolution image and the high resolution output of the previous frame.

3.2 Super-Resolution Surface Inference

Once the network has been trained, new low resolution input images are given to the network to infer the corresponding high resolution output images. For the inference step, we build upon the frame-recurrent neural network architecture proposed by Sajjadi *et al.*

[6]. At the current timestep t, the network is given the input I_t^{LR} and the previous high resolution prediction O_{t-1}^{est}, warped using the image-space flow, for temporal coherence. It produces the current prediction O_t^{est}, and after a post-processing step also the final color $C_t^{est} \in [0,1]^{3\times 4H\times 4W}$.

Figure 3 shows all data that are used and inferred by the network, together with the different processing stages an inference step is comprised of. These processing stages are:

1. Upscaling and Warping: After upscaling the screen-space flow F_t^{LR}, it is used as described to warp all previous estimated maps O_{t-1}^{est}, leading to $\tilde{O}_{t-1}^{est} \in \mathbb{R}^{6\times 4H\times 4H}$.

2. Flattening: Next, the warped previous maps $\tilde{O}_{t-1}^{est}$ are flattened into the low resolution by applying a space-to-depth transformation [6]

$$S_s : \mathbb{R}^{6\times 4H\times 4W} \rightarrow \mathbb{R}^{4^2 6\times H\times W}. \tag{1}$$

I.e., every 4×4 block of the high resolution image is mapped to a single pixel in the low resolution image. The channels of these $4\times 4 = 16$ pixels are concatenated, resulting in a new low resolution image O_f with 16-times the number of channels.

3. Super-Resolution: The super-resolution network then receives the current low resolution input I_t^{LR} (5 channels) and the flattened, warped prediction from the previous frame O_f (i.e., $16\cdot 6$ channels). The network then estimates the six channels of the output O_t^{est}, the high resolution mask, normal, depth, and AO maps.

4. Shading: To generate a color image, screen-space Phong shading with AO is applied as a post-processing step, i.e.,

$$C_{rgb} = \text{Phong}(c_a, c_d, c_s, c_m, N_t^{est}) * AO_t^{est}, \tag{2}$$

with the ambient color c_a, diffuse color c_d, specular color c_s and material color c_m as parameters.

The network also produces a high resolution mask M_t^{est} as output. While the input mask M_t^{LR} is comprised only of values -1 (outside) and +1 (inside), M_t^{est} can take on any value. Hence, M_t^{est} is clamped first to $[-1,+1]$ and then rescaled to $[0,1]$, leading to $M_t'^{est}$. This map shows a smooth fall-off of values across edges and and allows the network to smooth out edges via

$$C_t^{est} = \text{lerp}(c_{\text{bg}}, C_{rgb}, M_t'^{est}), \tag{3}$$

with c_{bg} being the background color.

3.3 Loss Functions

In the following, we describe the loss functions we have used during training to calculate the model error in the optimization process. Via the loss functions, the importance of certain features—and thus the fidelity by which they can be inferred—can be controlled. The single loss functions we use are common in artificial neural networks, yet in our case they are applied separately to different channels of the inferred and ground truth images. The total loss function used for training the network is a weighted sum of the loss functions below. In section 5, we analyze the effects of different loss functions on the reconstruction quality.

1. Spatial loss: As a baseline, we employ losses with regular vector norms, i.e. L_1 or L_2, on the different outputs of the network. Let X be either the mask M, the normal map N, the depth map D, the AO map AO or the shaded output C. Then the L_1 and L_2 losses are given by:

$$\mathscr{L}_{X,L_1} = ||X_t^{est} - X_t^{GT}||_1, \;\; \mathscr{L}_{X,L_2} = ||X_t^{est} - X_t^{GT}||_2^2.$$

2. Perceptual loss: Perceptual losses, as proposed by Gatys *et al.* [49], Dosovitskiy and Brox [50], and Johnson *et al.* [51] have been widely adopted to guide learning tasks towards detailed outputs instead of smoothed mean values. The idea is that two images are similar if they have similar activations in the latent space of a pre-trained network. Let ϕ be the function that extracts the layer activations when feeding the image into the feature network. Then the distance is computed by

$$\mathscr{L}_{X,P} = ||\phi(X_t^{est}) - \phi(X_t^{GT})||_2^2. \tag{4}$$

As feature network ϕ, the pretrained VGG-19 network [52] is used. We used all convolution layers in all spatial dimensions as features, with weights scaled so that each layer has the same average activation when evaluated over all input images.

Since the VGG network was trained to recognize objects in color images in the space $[0,1]^3$, the shaded output C can be directly used. This perceptual loss on the color space can be backpropagated to the network outputs, i.e. normals and ambient occlusions, with the help of the differentiable Phong shading. This shading is part of the loss function, and is implemented such that gradients can flow from the loss evaluation into the weight update of the neural network during training. Hence, with our architecture the network receives a gradient so that it can learn how the output, e.g., the generated normals, should be modified such that the shaded color matches the look of the target image. When applying the perceptual loss on other entries, the input has to be transformed first. The normal map is rescaled from scale $[-1,+1]^3$ to $[0,1]^3$, and depth and masking maps are converted to grayscale RGB images. We did not use additional texture or style loss terms [36], [49], since these introduce artificial details and roughness in the image which is not desired in smooth isosurface renderings.

3. Temporal loss: All previous loss functions worked only on the current image. To strengthen the temporal coherence and reduce flickering, we employ a temporal L_2 loss [53]. We penalize differences between the current high resolution image O_t^{est} and the previous, warped high resolution image $\tilde{O}_{t-1}^{est}$ with

$$\mathscr{L}_{X,\text{temp}} = ||X_t^{est} - \tilde{X}_{t-1}^{est}||_2^2, \tag{5}$$

where X can be M, N, D, AO or C.

In the literature, more sophisticated approaches to improve the temporal coherence are available, e.g. using temporal discriminators [40]. These architectures give impressive result, but are quite hard to train. We found that already with the proposed simple temporal loss, good temporal coherence can be achieved in our current application. We refer the readers to the accompanying video for a sequence of a reconstruction over time.

4. Loss masking: During screen-space shading in the post-process (see section 3.2), the output color is modulated with the mask indicating hits with surface points. Pixels where the mask is -1 are set to the background color. Hence, the normal and AO values produced by the network in these areas are irrelevant for the final color.

To reflect this in the loss function, loss terms that do not act on the mask (i.e. normals, ambient occlusions, colors) are itself modulated with the mask so that areas that are masked out don't contribute to the loss. We found this to be a crucial step that simplifies the network's task: In empty regions, the ground truth images are filled with default values in the non-mask channels, while with loss masking, the network does not have to match these values.

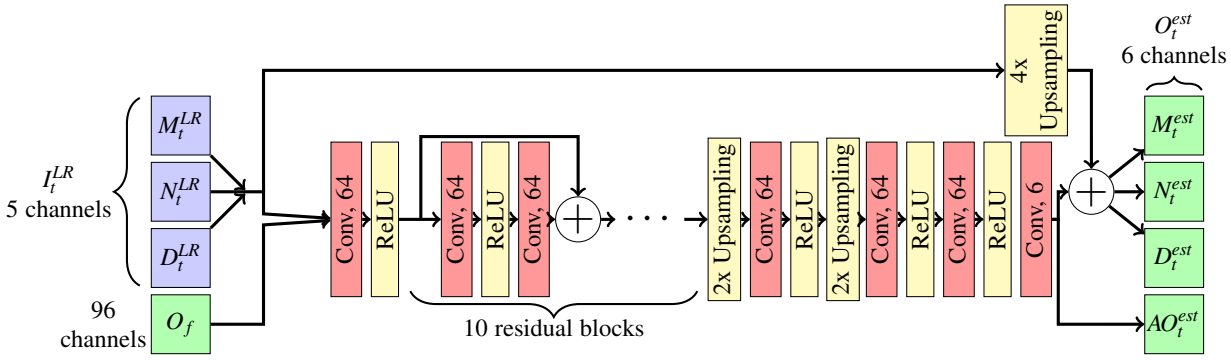


Fig. 4: Network architecture for the SRNet. Within the network, $\oplus$ indicates component-wise addition of the residual. All convolutions use 3x3 kernels with stride 1. Bilinear interpolation was used for the upsampling layers.

5. Adversarial Training: Lastly, we also employed an adversarial loss as inspired by Chu *et al.* [40]. In adversarial training, a discriminator network is trained parallel to the super-resolution network generator. The discriminator receives ground truth images and predicted images, and is trained to classify whether the input was ground truth or not. This discriminator is then used in the loss function of the generator network (see, e.g., Goodfellow *et al.* [54] for further details).

In our scenario, for evaluating the predicted images the discriminator is provided with

- the high resolution output O_t^{est}, and optionally the color C_t^{est},
- the input image I_t^{LR} as a conditional input to learn and penalize the mismatching between input and output,
- the previous frames $I_{t-1}^{LR}, O_{t-1}^{est}$, and optionally C_{t-1}^{est} to learn to penalize for temporal coherence.

To evaluate the discriminator score of the ground truth images, the predicted images O^{est} are replaced by O^{GT}.

As a loss function of the discriminator, we use the binary cross entropy loss. More concretely, let z be the input over all timesteps and $G(z)$ the generated results, i.e. the application of the super-resolution prediction on all timesteps. Let $D(x)$ be the discriminator that takes the high resolution outputs as input and produces a single scalar score. Then the discriminator is trained to distinguish fake from real structures by minimizing

$$\mathscr{L}_{\text{GAN},D} = -\log(D(x)) - \log(1 - D(G(z))). \tag{6}$$

The generator is trained to minimize

$$\mathscr{L}_{\text{GAN},G} = -\log(D(G(z))) \tag{7}$$

4 Learning Methodology

In this chapter, we provide a detailed description of the used network architecture, as well as the training and inference steps. We also shed light on the dependency of the reconstruction quality on the used loss functions.

4.1 Network Architecture

The network architecture is a fully convolutional frame-recurrent neural network (FRVSR-Net) consisting of a series of residual blocks [6]. An illustration of the network's building blocks and its topology is given in Figure 4. he modifications we have performed are with respect to the number of input and output channels, the other parts of the network are kept unchanged. The generator network starts with one convolutional layer to reduce the 101 input channels (5 from I_t^{LR}, $6*4^2$ from O_f) into 64 channels. Next, 10 residual blocks are used, each of which contains 2 convolutional layers. These are followed by two upscaling blocks ($2\times$ bilinear upscaling, a convolution and a ReLU) arriving at a $4\times$ resolution, still with 64 channels. In a final step, two convolutions process these channels to reduce the latent feature space to the desired 6 output channels. All layers use 3x3 kernels with stride 1.

The network is a residual network, i.e. it learns changes to the input. As shown in previous work [32], this improves the network's capability to generalize to new data, as it can focus on generating the residual content. Hence, the 5 channels of the input are bi-linearly upsampled and added to the first five channels of the output, producing M_t^{est}, N_t^{est} and D_t^{est}. The only exception is AO_t^{est}, which is inferred from scratch, as there is no low resolution input AO map.

4.2 Training Data

The training and validation data consists of images of isosurfaces from different timesteps and multi-resolution versions of the Ejecta dataset. This dataset stems from a particle-based simulation of a supernova that was resampled to a grid with resolutions from 256^3 to 1024^3. We choose this test suite because it contains isosurfaces showing many different geometric structures, ranging from very small details to rather smooth low-frequency parts. Of these, we rendered 500 sequences, each consisting of 10 frames. For each sequence, two views are selected at random and used as start view and end view of a smooth camera path. Eight additional in-between views are then computed along the path, and used to render corresponding frames at a low image resolution of 128^2, see Figure 5 for examples. 5000 sub-regions with a spatial size of 32^2 (for the low image resolution) and a temporal length of 10 are randomly cropped from the initial sequences, and are split into training (80%) and validation (20%) data. By rejection sampling we ensure that in each sub-region at least 50% of the pixels show a surface hit. The smaller spatial size is needed to fit multiple inputs at once into the memory during training and benefit from batch processing in the optimizer. The number of timesteps is kept the same. On a single Nvidia GeForce GTX 1080 Ti, the networks are trained for 100 to 500 epochs with training times from 3 to 18 hours, depending on the used cost function.

At this point, it is worth noting that the low resolution input images are directly generated by the raycaster. This is

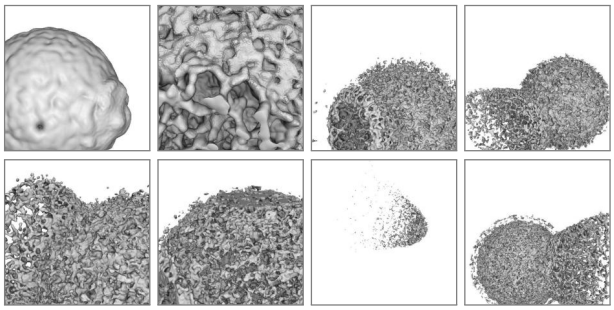


Fig. 5: Example images that are used to train our networks.

substantially different from common practice in image and video super-resolution [36], [40], where the inputs are low-pass filtered and downscaled versions of the original high resolution images. Thus, in our case the input images contain a huge amount of aliasing due to sub-sampling of the volumetric field, which poses a challenging task for the network.

The ground truth AO at a surface point is computed by sampling random directions on the hemisphere and testing for intersections between rays along these directions and the isosurface. This gives a much higher visual quality than screen-space AO, which tests samples against surface points based on screen-space depth.

To infer the current frame, the network takes the previous high resolution prediction O^{est}_{t-1} as input. Since the previous frame is not available in the first frame of a sequence, we evaluated different options to initialize the first previous high resolution input: Zeroing all entries, default setting to mask=0, normal=$[0,0,1]$, AO=1, and an upscaled version of the current input. Since we did not experience any noticeable difference between the three variants, we used the first and most simple option to train the network.

4.3 Loss Function Characteristics

The losses for training different super-resolution networks are obtained by using different weighted combinations of the individual losses described in subsection 3.3. Table 1 shows the specific weight combinations that are used for the networks we have analysed in our work.

For the networks trained with the losses in Table 1, Figure 6 shows a visual comparison of the surface structures (without AO) they infer from a given low resolution input image. In a number of tests we have confirmed the representativeness of these results, regardless of the type of isosurface and variations in the loss function weights. In subsection 5.2, we provide a quantitative evaluation that supports these findings.

All networks make use of a temporal loss $\mathscr{L}_{X,\text{temp}}$ to reduce flickering between successive frames. Due to the warping of the previous image, however, the use of a temporal loss can introduce smoothing. By changing the weighting between the temporal loss and the other losses, more focus can be put on either sharp details or improved temporal coherence.

As a baseline for comparison, we use a network that performs super-resolution on the low resolution color images, only including Phong shading but no AO. This network—"Shaded"—receives an RGB color image and a mask, and outputs the upscaled versions. In particular, this is different from our proposed upscaling process, where the inputs to the network comprise geometry, i.e., a depth and a normal field, and the final shading is performed in a deferred pass on the inferred high resolution normal field. Our experiments

Network	Losses
Shaded	$\mathscr{L}_{\text{GAN},G} + 0.5\mathscr{L}_{C,P} + 50\mathscr{L}_{C,\text{temp}}$
	network acts on shaded colors
L_1-color	$\mathscr{L}_{M,L_1} + \mathscr{L}_{AO,L_1} + 10\mathscr{L}_{C,L_1} + 0.1\mathscr{L}_{C,\text{temp}}$
L_1-geometry	$\mathscr{L}_{M,L_1} + \mathscr{L}_{AO,L_1} + 10\mathscr{L}_{N,L_1} + 100\mathscr{L}_{D,L_1} + 0.1\mathscr{L}_{C,\text{temp}}$
Perceptual	$\mathscr{L}_{M,L_1} + \mathscr{L}_{AO,L_1} + \mathscr{L}_{N,L_1} + \mathscr{L}_{D,L_1} + 0.1\mathscr{L}_{C,\text{temp}} + 5\mathscr{L}_{N,P} + \mathscr{L}_{AO,P}$
GAN	$\mathscr{L}_{M,L_1} + \mathscr{L}_{AO,L_1} + \mathscr{L}_{N,L_1} + \mathscr{L}_{D,L_1} + 0.1\mathscr{L}_{C,\text{temp}} + \mathscr{L}_{\text{GAN},G}$

TABLE 1: Networks and their specific loss function configurations.

have shown that the network "Shaded" infers the best results if it utilizes a combination of perceptual and adversarial loss. The exact loss function configuration is given in Table 1.

Figure 6, however shows that the "Shaded" network produces color distortions and over-blurring (see also Figure 2). The visual quality of this network falls consistently below the quality of the networks trained on geometry. This result supports our strategy to let the network learn upscaling the depth and normal fields, and shade the image in a deferred pass. The following networks all follow this strategy.

The second network we evaluate is "L_1-color", which is trained with L_1 loss on colors. It acts on depths and normals, yet it lets the loss function consider the colors after deferred shading. Apparently, this deteriorates the quality of the inferred output and produces rather washed out results.

The networks "L_1-geometry", "Perceptual' and "GAN" also train on depths and normals, yet they work with the mask, depth, normal and AO maps in the loss function. Thus, all three networks are capable of focusing more on the geometric properties of isosurfaces rather than their appearance. This is confirmed by our results, which show that these networks can infer far more details from the low resolution inputs.

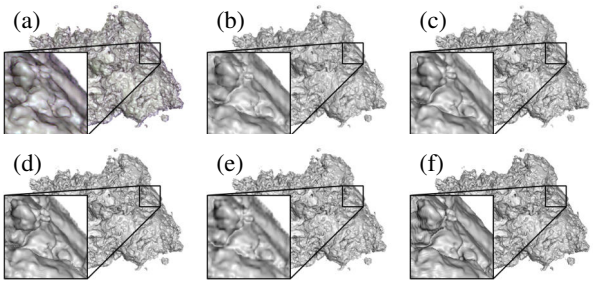


Fig. 6: Visual comparison of networks with different loss function configurations: (a) Shaded, (b) L_1-color, (c) L_1-geometry (our final model), (d) Perceptual, (e) GAN, (f) ground truth

The network "L_1-Geometry" trains only with L_1-losses on all input channels, i.e., mask, depth, normal, and AO. Adding a perceptual loss on the normal and AO fields, i.e., network "Perceptual", doesn't lead to any visual differences (Figure 6). We attribute this to the fact that the VGG-19 network was trained on color images and does not explicitly consider the relationships between geometric variations—given by the depth and normal fields—and the shaded output. We also noticed that the training time increases about a factor of six when using the VGG-19 network. Even though this drawback can probably be weakened by using fewer layers of the VGG-19 network, we still expect a significant performance loss if quality is maintained.

With network "GAN", we evaluate the influence of an adversarial loss on reconstruction fidelity. In our experiments, however, the "GAN" network produces more high-frequent details that actually decrease the quality of the reconstruction, and it significantly increases both training time and memory requirements by the discriminator. As a consequence, we decided to focus on directly supervised networks instead of GAN variants in this work.

In summary, of all tested networks the "L_1-geometry" network with minor objective on temporal coherence shows superior performance, both in terms of training time and reconstruction quality. This network does only see shaded colors in the temporal coherence loss during the training process, and is thus forced to focus primarily on the reconstruction of geometry.

5 Evaluation

We compare the "L_1-geometry" network with bi-linear and bi-cubic filtering as baseline methods on unseen test data. To validate how good the trained networks generalize to new isosurfaces, we let the network also upscale low resolution isosurface images of volumes which were never shown during training (see Figure 7): A CT scan of a human skull with a resolution of 256^3, a CT scan of a human thorax at 256^3, and a numerical simulation of a Richtmyer-Meshkov instability at 1024^3.

5.1 Qualitative Evaluation

To analyse the visual quality of network-based upsampling, we used the "L_1-geometry" network to upsample images of isosurfaces in the Ejecta dataset from perspectives that were never seen during training (see Figure 1 and Figure 7). The results indicate that the network can effectively infer the surface structures from the structures it has learned during training. In particular, compared to bi-linear and bi-cubic upsampling the network infers meaningful details in line with the geometric surface properties.

To demonstrate the reconstruction quality even for isosurfaces in datasets that were never seen by the network during training, we compare the results of bi-linear and network-based upscaling to the ground truth images using the datasets introduced before. The accompanying video shows the results when the network considers frame-to-frame coherence during animations. For the human skull, which exhibits smooth surfaces similar to Ejecta, the inference results are very close to the ground truth. A similar result is obtained when upscaling images of isosurfaces in the Richtmyer-Meshkov dataset. Despite the many fine-grained geometric details, the network can reconstruct the isosurface in a fairly accurate way. The network, however, faces difficulties when applied to images of an isosurface exhibiting geometric details at a higher frequency than it was trained on, as for example in the Thorax dataset. In this case, the network cannot reconstruct all fine-scale details accurately and rather blurs out the missing surface structures.

5.2 Quantitative Evaluation

To quantify the error that is introduced by learning-based isosurface reconstruction, all datasets are rendered 500 times from different views, and for each upscaled image the PSNR and SSIM are computed between the high resolution ground truth rendering and the reconstruction. The results, in the form of the medians, the $n\%$ quantiles, and the range of outliers, are shown in Figure 8 and Figure 9.

The PSNR is computed as

$$\mathrm{PSNR}(O_t^{est}, O_t^{GT}) = -10\log_{10}(||O_t^{est} - O_t^{GT}||_2^2), \tag{8}$$

where O_t^{est} and O_t^{GT} are the reconstructed and ground truth images, respectively. The SSIM is defined as

$$\mathrm{SSIM}(O_t^{est}, O_t^{GT}) = \frac{(2\mu_{est}\mu_{GT} + c_1)(2\sigma_{est,GT} + c_2)}{(\mu_{est}^2 + \mu_{GT}^2 + c_1)(\sigma_{est}^2 + \sigma_{GT}^2 + c_2)}, \tag{9}$$

where μ_{est} and μ_{GT} are the average values of O_t^{est} and O_t^{GT}, σ_{est}^2 and σ_{GT}^2 are the variances of O_t^{est} and O_t^{GT}, $\sigma_{est,GT}$ is the covariance between O_t^{est} and O_t^{GT}, and c_1 and c_2 are two small constants to avoid division by zero.

The first quantitative evaluation sheds light on the reconstruction quality of the networks that were trained using different loss functions. Figure 8 confirms the strength of the "L_1-geometry" network compared to the alternative variants presented in subsection 4.3. These results back up our decision to chose the "L_1-geometry" network as our preferred model, and to use it in the following quantitative analysis of the reconstruction quality.

In Figure 9, we analyze the reconstruction quality for both the normal and depth field, and further asses how well local illumination values and AO values can be inferred from the normal fields. Note here that AO values are not available at the low resolution input samples, because their simulation would be far too costly to maintain interactive frame rates. Thus, bi-linear and bi-cubic upscaling cannot generate high resolution AO. Therefore, the error measures were applied on the shaded color output, once without AO and including measures for bi-linear and bi-cubic, and once with AO and excluding bi-linear and bi-cubic upscaling. As can be seen, the PSNR and SSIM always slightly decrease when AO is added. This, however, is not particularly surprising, since one more quantity is inferred by the network that could introduce some error.

It can be seen that the "L_1-geometry" network always achieves better results than the other alternatives, even for isosurfaces of volumes that were not seen during training. This indicates the principal capability of neural networks to generalize to new data. Since the PSNR cannot capture the "sharpness" of the image very well [7], the differences are rather moderate when using the PSNR as quality metric. However, the differences become significant when using SSIM as quality metric. This is further confirmed by the images in subsection 5.1, which also show far better perceptual quality of network-based reconstruction compared to bi-linear and bi-cubic upscaling.

As described in subsection 4.2, the "L_1-geometry" network is trained solely on Ejecta (see Figure 5). This dataset does not contain the specific structures and rather smooth AO distribution observed in the Cloud dataset, a dataset of 12 clouds by Kallweit *et al.* [55]. An example of a volume typical for this dataset can be seen in Figure 10. We therefore re-trained the network for 600 epochs on images of isosurfaces in the Cloud dataset. The statistics in Figure 9 indicate that the re-trained network performs substantially better on the Cloud dataset, both in terms of PSNR and SSIM. This is also confirmed by Figure 10, which shows the inference results of the "L_1-geometry" network, once trained on Ejecta and once on Cloud. The comparison to the ground truth image indicates strong improvement of the reconstruction accuracy when specializing the network on a specific dataset.

The evaluation so far shows that network-based upscaling outperforms bi-linear and bi-cubic upscaling, yet it does not provide information about the number of erroneously inferred pixels and

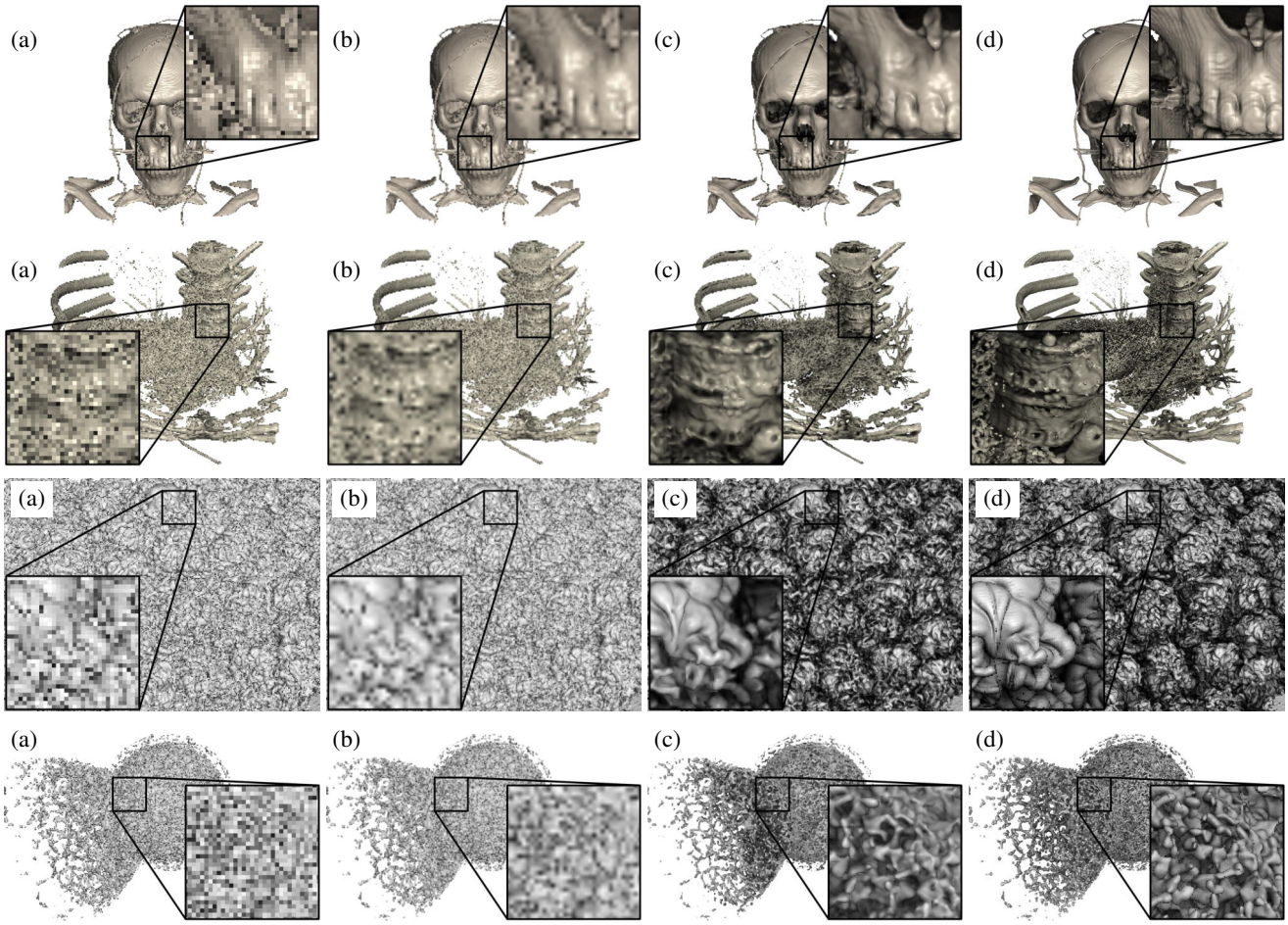


Fig. 7: Comparison of upscaling quality: (a) input, (b) bi-linear, (c) our network, (d) ground truth on the Skull, Thorax, Richtmyer-Meshkov and Ejecta dataset (top to bottom).

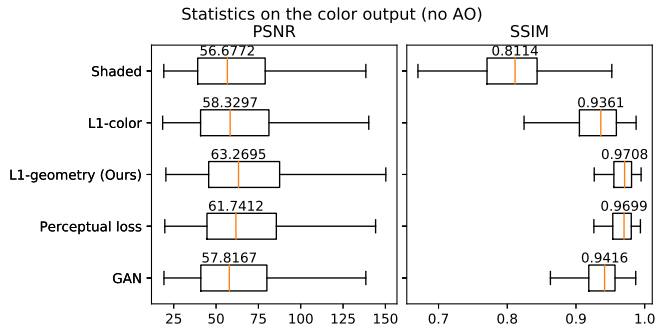


Fig. 8: Reconstruction quality of networks trained on the Cloud dataset using different loss functions. The orange line shows the median and the box outlines the 25% and 75% quantile.

their deviation from the ground truth. To investigate this aspect, we further analyze how many "good pixels" and "bad pixels" exist using the Regression Error Characteristic (REC) curves [56]. Given a certain error tolerance, REC curves show the percentage of pixels that are accurate according to the ground truth, i.e. $REC(\tau) = P(|x^{est} - x^{GT}| \leq \tau)$. It is widely used as a better performance description of a predictive model comparing to error statistics like PSNR, because the performance is illustrated across the range of errors. In our cases, we evaluate the REC curves on the normal, depth, AO and the gray-scale pixel intensity after shading without AO for our model in together with bi-linear and bi-cubic upscaling in Figure 11(a). First and foremost, the measurements indicate that erroneously inferred pixels cannot be avoided by any of the methods. The pixel-wise difference images in Figure 11(b) and (c) indicate that these errors are predominantly introduced along the silhouettes and cavities, where sometimes the network cannot accurately extrapolate the sharp transitions. Second, it is shown that network-based inference can reduce the number of pixels with a certain deviation significantly. With the Ejecta dataset as example, bi-linear upscaling generates an image in which 16% of the pixels have an absolute error to the ground truth intensity value larger than 0.1, with a maximal error of 1 between completely dark and completely bright. When using network-based upscaling, this number is reduced to 12.4%. Similar results hold for all other test datasets, demonstrating the superior quality of network-based upscaling.

5.3 Timings

To evaluate the performance of isosurface super-resolution, we compare it to volumetric ray-casting on the GPU using an empty-space acceleration structure. Rendering times for the shown isosurfaces in the four test datasets are given in Table 2, for a

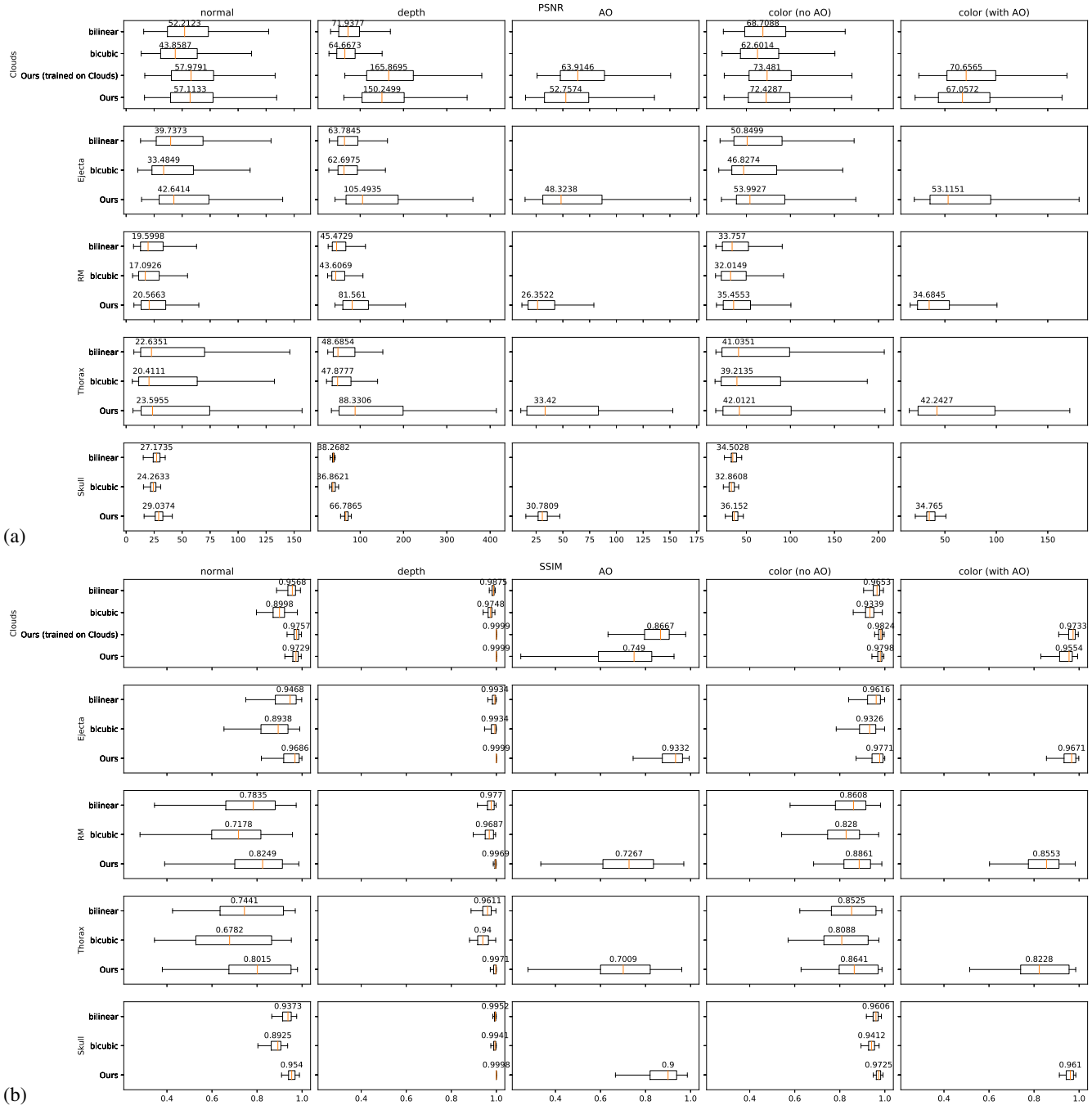
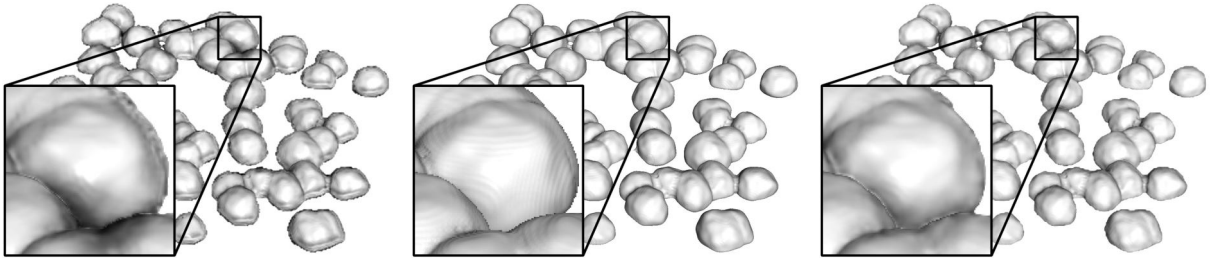


Fig. 9: (a) PSNR and (b) SSIM on the training and test examples for different upscaling approaches. Box plots show medians, quantiles and outliers. Since bi-linear and bi-cubic upsampling cannot infer AO, error measures are not available for these fields.

viewport size of 1920x1080. Each dataset was rendered from a number of different views along a pre-recorded path, so that the dataset covers the entire viewport. The isosurface renderer is implemented with Nvidia's GVDB library [57], an optimized GPU raytracer written in CUDA. The super-resolution network uses Pytorch. The timings were performed on a workstation running Windows 10, equipped with an Intel Xeon W-2123, 3.60Ghz, 8 logical cores, 64GB RAM, and a Nvidia RTX Titan GPU.

The table shows the time to render the ground truth image at full resolution with AO, the rendering times for the low resolution input without AO, and the time to perform super-resolution upscaling of the input using the "L_1-geometry" network. As the computation times for all three different quantities do not differ significantly from frame to frame, the average time is reported. The time to warp the previous image, perform screen-space shading, and IO between the renderer and the network are not included. For rendering the ground truth AO, 128 samples were taken. This gives reasonable results, but noise is still visible.

As one can see from Table 2, the time to simulate AO increases the computational cost significantly. Because the Ejecta dataset



(a) Network trained on w/o Clouds. (b) Ground truth rendering of Cloud. (c) Network re-trained only on Clouds.

Fig. 10: (a) Our network trained on the Ejecta dataset cannot faithfully reproduce the smooth geometry and silhouettes in the Cloud dataset, as shown in the ground truth rendering (b). It tends to "overshoot" the AO values and produces artefacts at the boundaries. (c) By re-training the network on images of isosurfaces in the Cloud dataset, the reconstruction quality is improved.

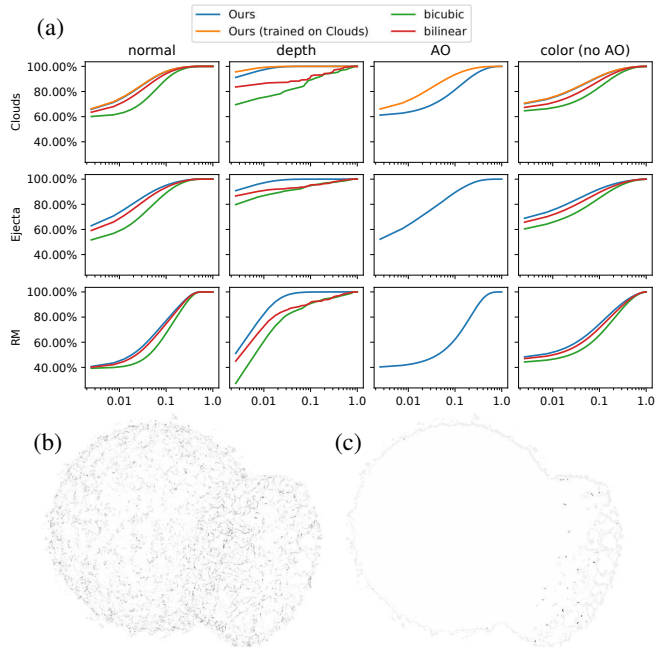


Fig. 11: (a) REC curves, the cumulative accuracy over the error tolerance on L_1-distance $|x^{est} - x^{GT}|$. A higher value is better. The spatial distribution of the pixel-wise error for the (b) normal and (c) depth map.

contains less empty blocks that can be skipped during rendering than the Richtmyer-Meshkov dataset, the computation time for the first hit (high resolution image without AO) is twice as high as that for the Richtmyer-Meshkov dataset. As expected, the time to evaluate the super-resolution network stays constant for all four datasets, as it only depends on the viewport size.

As an example, the total time to render the input and 4 times upscale images of isosurfaces in the Richtmyer-Meshkov dataset is $0.014s + 0.072s = 0.086s$. Hence, the network approximately takes the same time it takes to render the full resolution without AO ($0.088s$), but in this time also produces a smooth AO map. More prominently, for the Ejecta dataset, a high resolution rendering without AO takes $0.163s$, approximately 50% longer than rendering the low resolution version and upscaling it, which requires $0.103s$ in total. The latter version also provides AO "for free". Once AO is included in the high resolution rendering, the rendering time increases to $18.6s$, hence the super-resolution outperforms the high resolution renderer by two orders of magnitudes.

Dataset	High-res (no AO)	High-res (with AO)	Low-res	Super-res
Skull 256^3	0.057	4.2	0.0077	0.071
Thorax 256^3	0.069	9.1	0.010	0.071
R.-M. 1024^3	0.088	14.5	0.014	0.072
Ejecta 1024^3	0.163	18.6	0.031	0.072

TABLE 2: Timings in seconds for rendering an isosurface in FullHD (1920x1080) resolution, averaged over 10 frames.

6 Discussion

Our results demonstrate that deep learning-based inference has potential for upscaling tasks beyond classical image-based approaches. The trained network seems to infer well the geometric properties of isosurfaces in volumetric scalar fields. We believe this result is of theoretical interest on its own, and at the same time opens up new perspectives in a number of practical use cases. Even though it is not possible, in general, to predict the error that is introduced by the network, we believe there are two classes of applications were learning-based isosurface inference is eligible: Into the first class fall applications were a high resolution surface does not exist, for instance, because due to time and memory constraints only a low resolution volumetric field can be acquired. In such scenarios, learning-based inference might be able to predict where certain features can occur and, thus, can guide refined simulations or measurements. Regarding this use case, it will be important to investigate whether artificial neural networks can infer from a given high resolution image of an isosurface in a low resolution volume the isosurface rendering from the corresponding high resolution volume.

Into the second class fall applications were the user is willing to make tradeoffs in fidelity and speed, i.e., were reconstruction quality can be sacrificed for speed, at least temporarily or locally. In the following, we shed light on two such applications and demonstrate the practical usefulness of isosurface inference.

6.1 Use Cases

One application is the utilization of isosurface inference to support an interactive exploration of high resolution volume datasets. When the high resolution dataset cannot be rendered at interactive rates, it is common practice in many visualization tools to render a low

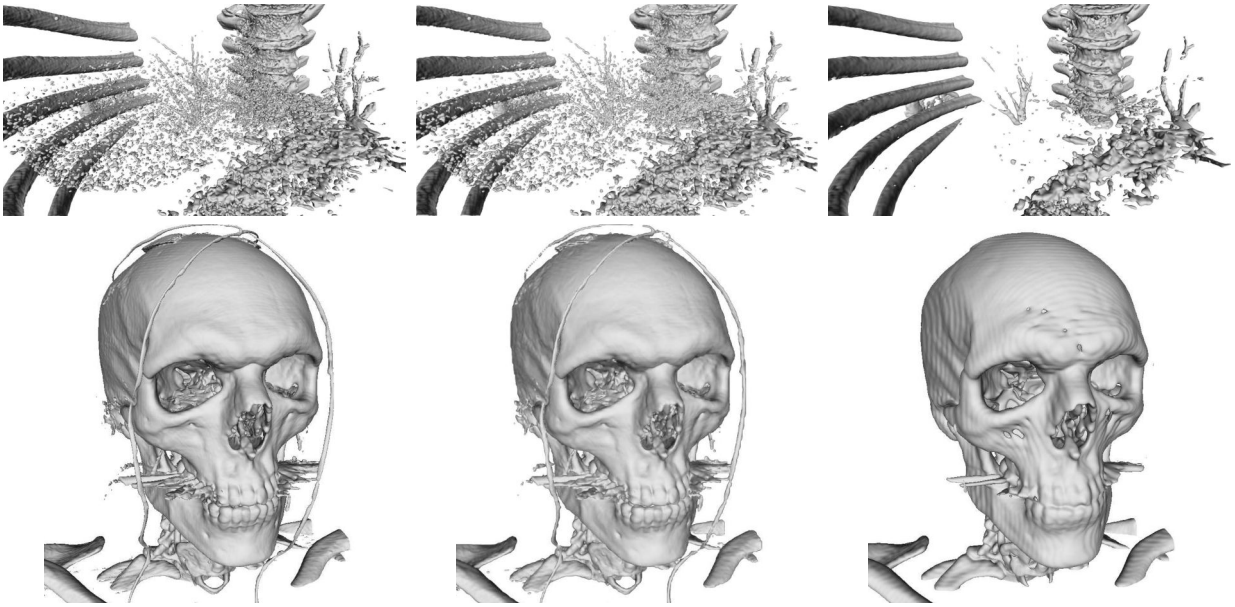


Fig. 12: Level of detail rendering for interactive exploration. Left: Rendering of isosurfaces in the original datasets at full image resolution. Middle: Rendering the isosurfaces in the original datasets at $1/4$ the image resolution and upscaling to full image resolution. Right: Rendering the isosurfaces in the low-pass filtered and down-sampled datasets with half the resolution in each dimension at full image resolution.

resolution version at full image resolution during interaction, and to switch back to the high resolution dataset once the camera stands still. This enables an interactive exploration, yet provides views in which many details are lost and even the topology of the isosurface can be corrupted due to the downsampling process that is used to generate the low resolution version (see 3rd column in Figure 12). In contrast, upscaling a low resolution image of the high resolution dataset (2nd column in Figure 12) generates a far more detailed view and preserves topology to a large extend. Notably, upscaling the low resolution image of the high resolution isosurface requires roughly the same time as rendering the low resolution isosurface at full image resolution.

The same principle can be applied in remote visualization, where in practice the bandwidth of the communication channel across which rendered images are transmitted often limits the streaming performance. Thus, the degree of interactivity often falls below what a user expects. To weaken this limitation, during interaction low resolution images of the dataset can be streamed to the client-side and upscaled using trained networks.

As a second use case we have integrated isosurface inference into foveated rendering. In foveated rendering, a focus region in the image is given by the falloff of acuity in the visual periphery. Since fine details can only be sensed within a small portion (5°) of the visual field, with increasing angular distance from the central region of visual stimulus, the number of samples can be reduced accordingly. This is exploited in foveated rendering to reduce the number of samples rendered in the peripheral region, by either upscaling low resolution renderings [58] or interpolating between a sparse set of initial samples in this region [59].

To use network-based inference in foveated rendering, we utilize renderings at different image resolution. While the image is rendered at full resolution in the region of highest acuity, it is rendered at $1/4$ this resolution in the exterior (taking over the full resolution samples in the focus region) and upscaled by the super-resolution network. The two images are then smoothly blended together. As shown in Figure 13a,b, the difference between the full resolution rendering in the focus regions and the upscaled low resolution version is almost indistinguishable, yet a significant lower number of samples is required to generate the final image. This number can be further reduced by generating and blending multiple images at ever lower resolution, i.e., by rendering images using $1/2$ and $1/4$ the full resolution and using networks to upscale to the full resolution.

In foveated rendering, the reconstruction error that may be introduced by the network is fully acceptable, since it only occurs in the region outside the users central region of visual stimulus. Yet as we have shown in this work, learning-based upscaling can far better maintain geometric and topological features than other techniques like bi-linear upscaling. Thus, transitions between multiple resolutions are far less pronounced and can be removed more effectively.

6.2 Conclusion and Future Work

We have introduced and analyzed a deep learning technique for isosurface super-resolution with AO. The proposed recurrent network architecture with temporally coherent adversarial training makes it possible to infer highly detailed images from low resolution input renderings. The network reconstructs high resolution images of isosurfaces including ambient occlusions at a performance that is significantly faster than that of an optimized ray-caster at full resolution. We have published the source code for training and inference as well as the trained networks and datasets on https://github.com/shamanDevel/IsosurfaceSuperresolution.

The quality of upscaling seems to indicate that not a specific isosurface is learned, but rather that the network is able to generalize, i.e., to infer the geometric properties of isosurfaces in volumetric scalar fields. Yet especially when the network's learned

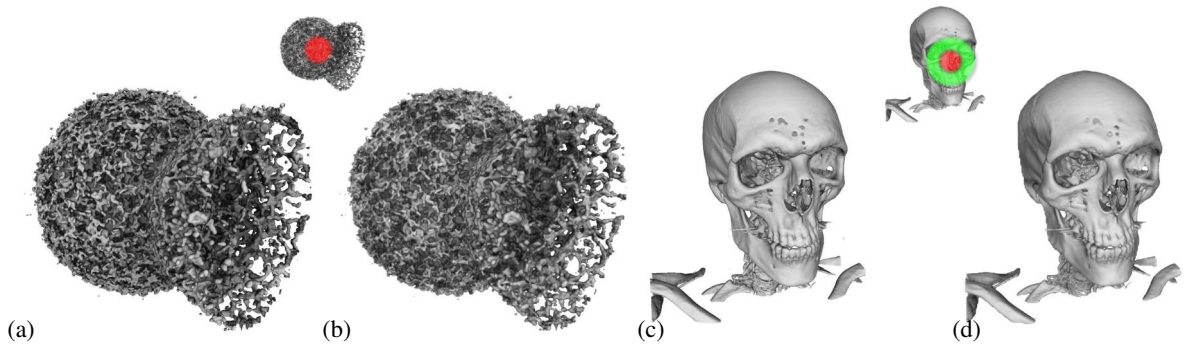


Fig. 13: Foveated rendering. Rendering of the original datasets in (a) and (c). (b) In the region of highest acuity (indicated by the red circle in the inlay), the isosurface is sampled at full resolution and smoothly blended with an upscaled image at $1/4$ the resolution. (d) Same as (b), but region of highest acuity is decreased and blended with upscaled images at $1/2$ (green area) and $1/4$ (exterior) the resolution. In (b) and (d), respectively, 16% and 11% of the samples in (a) and (c) are used.

representation is not sufficient due to limited training sample variation, network-based reconstruction can lead to distortions in the inferred structures. We have demonstrated that this can be counteracted by specializing the network on certain types of isosurfaces, such as they occur in certain types of simulated or measured physical fields. The quality of the reconstruction can be substantially improved if the network is given the chance to see the type of isosurface it is used to infer. Nevertheless, it is arguable that network-based inference should be used carefully in applications were highest accuracy is required, e.g., in medical imaging.

On the other hand, we have shown applications where a reconstruction error is tolerable, e.g., during interactive navigation in large datasets and in foveated rendering. In such applications, network-based inference provides a very effective means to balance between reconstruction quality and performance.

The proposed method only represents a first step towards learning-based data inference, and we see numerous promising and interesting avenues for future research. Among others, it will be important to analyze how sparse the input data can be so that a network can still infer on the geometry of the underlying structures. Furthermore, we will shed light on the inference of additional rendering effects such as soft shadows. Finally, we will investigate the extension of our approach to support transparency and multiple-scattering effects, by going beyond image-based inference and integrating volumetric representations in the training and inference steps.

Acknowledgments

This work is supported by the ERC Starting Grant *realFlow* (StG-2015-637014).

References

[1] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[2] A. Kappeler, S. Yoo, Q. Dai, and A. K. Katsaggelos, "Video super-resolution with convolutional neural networks," *IEEE Transactions on Computational Imaging*, vol. 2, no. 2, pp. 109–122, 2016.

[3] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin, "Volume upscaling with convolutional neural networks," in *Proceedings of the Computer Graphics International Conference*, ser. CGI '17. New York, NY, USA: ACM, 2017, pp. 38:1–38:6. [Online]. Available: http://doi.acm.org/10.1145/3095140.3095178

[4] J. Han and C. Wang, "Tsr-tvd: Temporal super-resolution for time-varying data analysis and visualization," *IEEE Transactions on Visualization and Computer Graphics (to appear)*, 2019.

[5] 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 (to appear)*, 2019.

[6] M. S. Sajjadi, R. Vemulapalli, and M. Brown, "Frame-recurrent video super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6626–6634.

[7] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[8] M. Levoy, "Display of surfaces from volume data," *IEEE Comput. Graph. Appl.*, vol. 8, no. 3, pp. 29–37, May 1988. [Online]. Available: https://doi.org/10.1109/38.511

[9] J. Danskin and P. Hanrahan, "Fast algorithms for volume ray tracing," in *Proceedings of the 1992 Workshop on Volume Visualization*, ser. VVS '92. New York, NY, USA: ACM, 1992, pp. 91–98. [Online]. Available: http://doi.acm.org/10.1145/147130.147155

[10] L. M. Sobierajski and A. E. Kaufman, "Volumetric ray tracing," in *Proceedings of the 1994 Symposium on Volume Visualization*, ser. VVS '94. New York, NY, USA: ACM, 1994, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/197938.197949

[11] M. Wan, A. Kaufman, and S. Bryson, "High performance presence-accelerated ray casting," in *Proceedings of the conference on Visualization '99*, 1999, pp. 379–386.

[12] M. Sramek, "Fast surface rendering from raster data by voxel traversal using chessboard distance," in *Proceedings Visualization '94*, Oct 1994, pp. 188–195.

[13] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister, "Sparseleap: Efficient empty space skipping for large-scale volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 974–983, Jan 2018.

[14] J. Kruger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, ser. VIS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 38–. [Online]. Available: https://doi.org/10.1109/VIS.2003.10001

[15] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross, "Real-time ray-casting and advanced shading of discrete isosurfaces," *Comput. Graph. Forum*, vol. 24, pp. 303–312, 2005.

[16] T. Klein, M. Strengert, S. Stegmaier, and T. Ertl, "Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware," in *IN: PROCEEDINGS OF IEEE VISUALIZATION '05*. IEEE, 2005, pp. 223–230.

[17] C. Braley, R. Hagan, Y. Cao, and D. Gračanin, "Gpu accelerated isosurface volume rendering using depth-based coherence," in *ACM SIGGRAPH ASIA 2009 Posters*, 2009, pp. 42:1–42:1.

[18] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián, "A single-pass gpu ray casting framework for interactive out-of-core rendering of massive volumetric datasets," *The Visual Computer*, vol. 24, no. 7, pp. 797–806, Jul 2008. [Online]. Available: https://doi.org/10.1007/s00371-008-0261-9

[19] M. Treib, K. Bürger, F. Reichl, C. Meneveau, A. Szalay, and R. Westermann, "Turbulence visualization at the terascale on desktop pcs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2169–2177, Dec 2012.

[20] F. Reichl, M. G. Chajdas, K. Bürger, and R. Westermann, "Hybrid Sample-based Surface Rendering," in *Vision, Modeling and Visualization*,

M. Goesele, T. Grosch, H. Theisel, K. Toennies, and B. Preim, Eds. The Eurographics Association, 2012.

[21] T. Fogal, A. Schiewe, and J. Kruger, "An analysis of scalable gpu-based ray-guided volume rendering," in *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, vol. 2013, 10 2013, pp. 43–51.

[22] J. Beyer, M. Hadwiger, and H. Pfister, "State-of-the-art in gpu-based large-scale volume visualization," *Computer Graphics Forum*, vol. 34, no. 8, pp. 13–37, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12605

[23] S. Zhukov, A. Iones, and G. Kronin, "An ambient light illumination model," in *Rendering Techniques '98*, G. Drettakis and N. Max, Eds. Vienna: Springer Vienna, 1998, pp. 45–55.

[24] E. Penner and R. Mitchell, "Isosurface ambient occlusion and soft shadows with filterable occlusion maps," in *Proceedings of the Fifth Eurographics / IEEE VGTC Conference on Point-Based Graphics*, ser. SPBG'08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 57–64. [Online]. Available: http://dx.doi.org/10.2312/VG/VG-PBG08/057-064

[25] F. Hernell, P. Ljung, and A. Ynnerman, "Efficient ambient and emissive tissue illumination using local occlusion in multiresolution volume rendering," in *Proceedings of the Sixth Eurographics / Ieee VGTC Conference on Volume Graphics*, ser. VG'07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 1–8. [Online]. Available: http://dx.doi.org/10.2312/VG/VG07/001-008

[26] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, and K. Hinrichs, "Interactive volume rendering with dynamic ambient occlusion and color bleeding," *Computer Graphics Forum*, vol. 27, no. 2, pp. 567–576, 2008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01154.x

[27] D. Jönsson, E. Sundén, A. Ynnerman, and T. Ropinski, "A survey of volumetric illumination techniques for interactive volume rendering," *Comput. Graph. Forum*, vol. 33, pp. 27–51, 2014.

[28] M. Mittring, "Finding next gen: Cryengine 2," in *ACM SIGGRAPH 2007 Courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007, pp. 97–121. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281671

[29] L. Bavoil, M. Sainz, and R. Dimitrov, "Image-space horizon-based ambient occlusion," in *ACM SIGGRAPH 2008 Talks*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 22:1–22:1. [Online]. Available: http://doi.acm.org/10.1145/1401032.1401061

[30] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[31] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep laplacian pyramid networks for fast and accurate superresolution," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2017, p. 5.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[35] T. Tong, G. Li, X. Liu, and Q. Gao, "Image super-resolution using dense skip connections," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4799–4807.

[36] M. S. Sajjadi, B. Scholkopf, and M. Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4491–4500.

[37] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia, "Detail-revealing deep video super-resolution," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[38] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, and T. Huang, "Robust video super-resolution with learned temporal dynamics," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2526–2534.

[39] Y. Jo, S. W. Oh, J. Kang, and S. J. Kim, "Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3224–3232.

[40] M. Chu, Y. Xie, L. Leal-Taixé, and N. Thuerey, "Temporally coherent gans for video super-resolution (tecogan)," *arXiv preprint arXiv:1811.09393*, 2018.

[41] R. Liao, X. Tao, R. Li, Z. Ma, and J. Jia, "Video super-resolution via deep draft-ensemble learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 531–539.

[42] Y. Xie, E. Franz, M. Chu, and N. Thuerey, "tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," *ACM Trans. Graph.*, vol. 37, no. 4, 2018.

[43] C. R. Alla Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila, "Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder," *ACM Transactions on Graphics*, vol. 36, pp. 1–12, 07 2017.

[44] M. Mara, M. McGuire, B. Bitterli, and W. Jarosz, "An efficient denoising algorithm for global illumination," in *Proceedings of High Performance Graphics*. New York, NY, USA: ACM, jul 2017.

[45] O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, and T. Ritschel, "Deep shading: Convolutional neural networks for screen space shading," *Comput. Graph. Forum*, vol. 36, no. 4, pp. 65–78, Jul. 2017. [Online]. Available: https://doi.org/10.1111/cgf.13225

[46] M. Berger, J. Li, and J. A. Levine, "A generative model for volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, pp. 1636–1650, 2017.

[47] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.

[48] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[49] 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*, 2016, pp. 2414–2423.

[50] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *Advances in neural information processing systems*, 2016, pp. 658–666.

[51] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*. Springer, 2016, pp. 694–711.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[53] D. Chen, J. Liao, L. Yuan, N. Yu, and G. Hua, "Coherent online video style transfer," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1105–1114.

[54] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[55] S. Kallweit, T. Müller, B. McWilliams, M. Gross, and J. Novák, "Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks," *ACM Trans. Graph. (Proc. of Siggraph Asia)*, vol. 36, no. 6, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130880

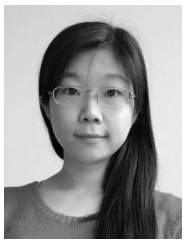
[56] J. Bi and K. P. Bennett, "Regression error characteristic curves," in *ICML*, 2003.

[57] R. K. Hoetzlein, "GVDB: Raytracing Sparse Voxel Database Structures on the GPU," in *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*, U. Assarsson and W. Hunt, Eds. The Eurographics Association, 2016.

[58] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3d graphics," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 164, 2012.

[59] M. Stengel, S. Grogorick, M. Eisemann, and M. A. Magnor, "Adaptive image-space sampling for gaze-contingent real-time rendering," *Comput. Graph. Forum*, vol. 35, pp. 129–139, 2016.

Sebastian Weiss received the M.Sc. degree from the Technical University of Munich in 2018. Currently, he is a doctoral student of computer science, Technical University of Munich. His research interests include volume visualization, deep learning and high performance GPU programming.



Mengyu Chu received the M.Eng. degree from Zhejiang University, China in 2014. Currently, she is a doctoral student of computer science, Technical University of Munich. Her research interests include fluid simulations and deep learning.

Nils Thuerey is an Associate-Professor at the Technical University of Munich (TUM). He works in the field of computer graphics, where a central theme of his research are physics simulations and deep learning algorithms. He received a tech-Oscar from the AMPAS in 2013 for his research on controllable smoke effects. He worked for three years as a post-doc at ETH Zurich and as R&D lead at ScanlineVFX, before starting at TUM in October 2013.

Rüdiger Westermann studied computer science at the Technical University Darmstadt and received his Ph.D. in computer science from the University of Dortmund, both in Germany. In 2002, he was appointed the chair of Computer Graphics and Visualization at TUM. His research interests include scalable data visualization and simulation algorithms, GPU computing, real-time rendering of large data, and uncertainty visualization.

Learning Adaptive Sampling and Reconstruction for Volume Visualization

Sebastian Weiss, Mustafa Işık, Justus Thies, and Rüdiger Westermann

Abstract—A central challenge in data visualization is to understand which data samples are required to generate an image of a data set in which the relevant information is encoded. In this work, we make a first step towards answering the question of whether an artificial neural network can predict where to sample the data with higher or lower density, by learning of correspondences between the data, the sampling patterns and the generated images. We introduce a novel neural rendering pipeline, which is trained end-to-end to generate a sparse adaptive sampling structure from a given low-resolution input image, and reconstructs a high-resolution image from the sparse set of samples. For the first time, to the best of our knowledge, we demonstrate that the selection of structures that are relevant for the final visual representation can be jointly learned together with the reconstruction of this representation from these structures. Therefore, we introduce differentiable sampling and reconstruction stages, which can leverage back-propagation based on supervised losses solely on the final image. We shed light on the adaptive sampling patterns generated by the network pipeline and analyze its use for volume visualization including isosurface and direct volume rendering.

Index Terms—Volume visualization, adaptive sampling, deep learning.

1 Introduction

WHICH are the data samples that are needed to generate an image of a data set that conveys the relevant information encoded in this data? This question is fundamental to data visualization since it asks for the importance of data samples from a perceptual point of view, rather than a signal processing standpoint that argues in terms of numerical accuracy.

Recent works in visualization have shown that artificial neural networks can perform an accurate reconstruction from a reduced set of data samples, by learning the relationships between a sparse, yet regular input sampling and the high-resolution output. Learned representations are then applied in the reconstruction process to infer missing data samples. This type of reconstruction has been performed in the visualization image domain to infer high-resolution images from given low-resolution images of isosurfaces [60], in the spatial domain to infer a higher resolution version of a 3D data set from a low-resolution version [64], and in the temporal domain to infer a temporally dense volume sequence from a sparse temporal sequence [21].

Others have proposed neural networks that are trained end-to-end to learn directly the visual data representations instead of the data itself. Berger *et al.* [3] propose a deep image synthesis approach to assist transfer function design, by letting an artificial neural network synthesize new volume-rendered images from only a selected viewpoint and a transfer function. He *et al.* [23] demonstrate that artificial neural networks can even be used to bridge the data entirely, by learning the relationships between the input parameters of a simulation and visualizations of the simulation results. Both approaches do not make any explicit assumptions about the relevance of certain structures in the data, yet the learned relationships between parameters and visual representations are considered in the image generation process.

- *All authors are with Technical University of Munich, Germany. E-mail: {sebastian13.weiss,m.isik, justus.thies, westermann}@tum.de.*

1.1 Contribution

Our goal is to make a further step towards learning visual representations of volumetric data sets, i.e. images of a volume when displayed in some form, by investigating whether a neural network can a) learn the relevance of structures for generating such representations, b) use this knowledge to adaptively sample a visual representation of a volumetric object, and c) reconstruct an accurate image from the sparse set of samples. Notably, even we can demonstrate for large volumes and image sizes that adaptive sampling can save rendering time, performance improvement is not our main objective. It is even fair to say that an optimized GPU volume ray-caster can hardly be beaten performance-wise. Our main objective is to gain an improved understanding of the learning skills of neural networks for generating visual representations in an unsupervised manner, by letting networks learn the relevance of certain structures for generating such representations. It can eventually become possible to generate data representations that compactly encode relevant structures in a way that can be used by a neural network to visualize the data. Such insights can further facilitate the use of transfer learning to construct synthetic data sets that contain the structures that are important for successful learning tasks on real data. For viewpoint selection, a network might learn to recommend views showing many important structures, and for training this information can be used to acquire more data from similar views.

To address our objectives, we introduce a novel network pipeline that is trained end-to-end to learn the relevance of certain structures in the data for generating a visual representation (Figure 1). This pipeline is comprised of two consecutive internal network stages: An importance network and a reconstruction network. Both networks work in tandem, in that the first learns to place samples along relevant structures by using the second network to give feedback on how well a visual representation of the data can be reconstructed from the sparse sampling. Our approach differs from previous adaptive sampling approaches in

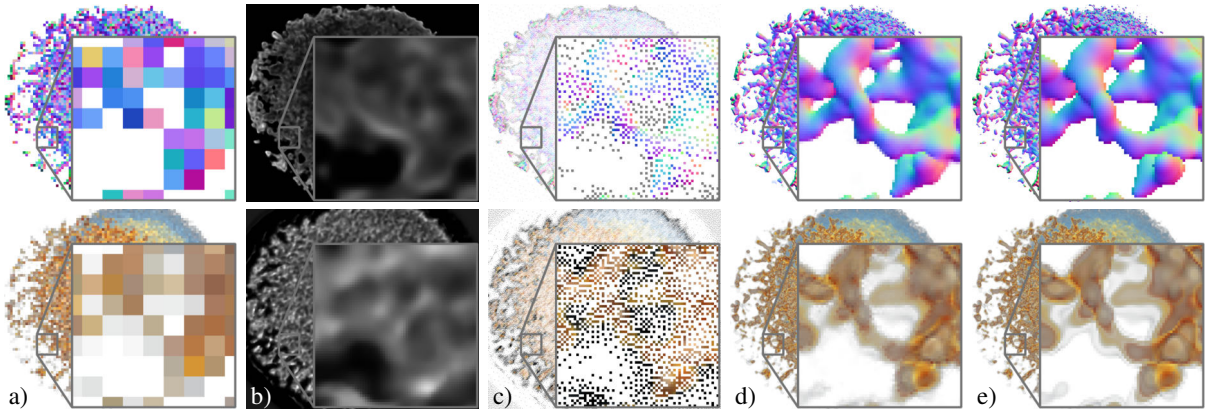


Fig. 1: An importance network, together with a differentiable sampler and a reconstruction network, takes a low resolution visualization (a) and infers an importance map (b) from it. From this map, an adaptive sampling pattern with adjustable number of samples (5% for iso, top; 10% for dvr, bottom) is derived, and a volume ray-caster samples the data according to these samples (c). The reconstruction network completes the visual representation from the sparse set of samples (d). The ground truth visualizations are shown in (e). The proposed network pipeline works on images of iso-surfaces (top) and direct volume renderings (bottom).

volume visualization [37, 31, 2] in that it does not rely on any specific saliency model to determine the image regions that need to be refined. In contrast, we propose a network-based processing pipeline that simultaneously learns where to sample and how to accurately reconstruct an image from the sparse samples, solely using losses on the reconstructed images.

For learning an importance map from a low-resolution visualization and reconstructing an image from a sparse set of pixel values, we use two modified versions of an EnhanceNet [53]. To enable network-based learning using gradient descent, two novel processing stages are introduced:

- A differentiable sampling stage that models the relationship between sample positions and visual representation.
- A differentiable image reconstruction stage using the pull-push algorithm [15, 32] to model the relationship between a sparse set of image samples and the reconstructed image.

In several experiments, we demonstrate that the importance network effectively selects structures that are relevant to the final visual representation. We focus on adaptive sampling in image-space, i.e., using surface samples and samples resulting from direct volume rendering. As a future direction of research, we outline adaptive sampling in object space, i.e., using data samples along view-rays. Our experiments include qualitative and quantitative evaluations, which indicate good reconstruction accuracy even from a few samples. The source code of our processing pipeline is available at https://github.com/shamanDevel/AdaptiveSampling, including some of the data sets that have been used for training and validation.

2 Related Work

In the following, we review previous works that share similarities with our approach from the fields of adaptive sampling for rendering as well as neural network-based image and volume reconstruction.

Adaptive Sampling for Rendering Adaptive rendering has a long tradition in computer graphics, to reduce the number of rays to trace against the scene and perform rasterization at lower image resolution. At the core of such approaches is the computation of importance values to steer the adaptive refinement, for instance, based on perceptual models [5, 42, 48], image saliency models using pixel variance [44, 49], image difference operations [40], or entropy-based measures [61], to name just a few. In the context of foveated rendering [16], where usually a static adaptive sampling pattern is used that moves with the users' gaze, a luminance-contrast-aware criterion was introduced to enable feature-aware adaptivity [58]. The importance map generation process is often started from an image preview that is calculated using a low resolution render pass or a high-resolution estimate that can be created in a significantly faster way than the final image.

For volume rendering, several approaches have investigated adaptive sampling in object space, to reduce the number of samples along the view rays [43, 9, 38, 6]. Adaptive image-space refinement has been proposed by Levoy [37], by using the color variances between pixels at low image resolution to decide whether to refine the image resolution locally. Kratz *et al.* [31] propose to use the difference image between two coarser resolution images, and locally refine where high differences are observed. Belayev *et al.* [2] render low-resolution images of isosurfaces and refine depending on how many pixels surrounding a pixel in the low-resolution view fulfill certain requirements. Frey *et al.* [13] use a fixed random sampling structure that is applied in a hierarchical manner to progressively refine the image.

The major differences between these approaches and our proposed sampling pipeline are as follows: Firstly, the pipeline learns to adapt the sampling in an unsupervised manner. A specific feature descriptor that steers the placement of samples is not used, and importance values are learned solely using losses on the reconstructed image. Secondly, the number of samples can be prescribed, which is not easily possible with existing schemes due to their pixel-iterative nature. Thirdly, the pipeline learns simultaneously the adaptive sampling and the image reconstruction from the sparse set of samples. In all previous schemes, the final interpolation step is decoupled from the sampling process.

Deep Learning for Upscaling and Denoising In recent years, deep learning approaches have been used successfully for single-image and video super-resolution tasks [11, 54, 55, 56, 52, 7], i.e., the upscaling of images and videos from a lower to some higher resolution. Many previous works let the networks learn to optimize for losses between the inferred and ground-truth images based on direct vector norms [29, 27]. GANs were introduced to prevent the undesirable smoothing of direct loss formulations [53,

36], and instead use a second network that discriminates real from generated samples and guides the generator. Convolutional architectures [11] with residual blocks [22] are popular generator architectures that offer training stability as well as high-quality inference. Losses based on the feature-space differences of image classification networks, e.g., a pre-trained VGG network [26], have shown to mimic well the human's capability to assess the perceptual similarity between two images.

The approach closest to ours is by Kuznetsov *et al.* [34] for learning adaptivity in Monte-Carlo path-tracing and denoising of the final image. A first network learns to adapt the number of additional paths from an initial image at the target resolution, which is generated via one path per pixel. A second denoising network learns to model the relationship between an image with increased variance in the color samples to the ground truth rendering [47, 41]. Conceptually, our approach differs in that it works on a low-resolution input map and then learns to freely position the sample locations in image space, i.e. it learns to place zero or one sample per pixel. This requires a completely different differentiable sampling stage, as well as a differentiable image reconstruction stage that can work on a sparse set of samples. Furthermore, Kuznetsov *et al.* use finite differences between images of different sample counts for gradient estimation. Incurring noise is reduced by averaging multiple samples with different sample counts, which is not possible in our approach where at most one sample per pixel is taken. Instead, we propose a sigmoid approximation that can be differentiated analytically.

In visualization, Zhou *et al.* [64] presented a CNN-based solution that upscales a volumetric data set using three hidden layers designed for feature extraction, non-linear mapping, and reconstruction, respectively. Han *et al.* [20] introduced a two-stage approach for vector field reconstruction via deep learning, by refining a low-resolution vector field from a set of streamlines. Berger *et al.* [3] proposed a deep image synthesis approach to assist transfer function design using GANs, by letting a network synthesis new volume-rendered images from only a selected viewpoint and a transfer function. The use of neural network-based inference of data samples in the context of in situ visualization was demonstrated by Han and Wang [21], where a network learns to infer missing time steps between 3D simulation results. He *et al.* [23] use neural networks for parameter-space exploration, by training a network to learn the dependencies between visual mappings of simulation results and the input parameters of the simulation. Guo *et al.* [17] designed a deep learning framework that produces coherent spatial super-resolution of 3D vector field data. Weiss *et al.* [60] extent image upscaling to geometry images of isosurfaces including depth and normal information. Instead of data upscaling, Tkachev *et al.* [57] predict a next time-step of a simulation and identify regions of interest by high variance between the network prediction and the ground truth. Common to all these approaches is the use of a regular sampling structure that does not consider the importance of samples in the inference step.

3 Learning to Sample

In the following, we discuss how the importance network makes use of both the adaptive sampling stage and the reconstruction network to learn where to place samples with higher density. The importance network (Subsection 3.1) receives an image of the data set at low resolution. This image L is of shape $C \times fH \times fW$, where W and H denote the screen resolution, and f the downsampling factor. This factor is set to $1/8$ in all of our experiments. Each image pixel is comprised of C channels, such as color, depth, and normal, representing what is seen through that pixel. The network is trained to learn an importance function $\mathscr{N}_I$ that generates a gray-scale importance map $I \in [0,\infty)^{H\times W}$ in which low and high values, respectively, indicate where less or more samples are taken.

The sampler $\mathscr{S}$ (Subsection 3.2) takes the importance map and places a given number of samples, e.g., 5% of the pixel, in the full resolution image $S \in \mathbb{R}^{C\times H\times W}$ according to the importance information. Only at these samples, the object is rendered. The reconstruction network learns a function $\mathscr{N}_R$ (Subsection 3.3) that reconstructs the final output $O \in \mathbb{R}^{C\times H\times W}$ from the sparse set of samples. We make the sampler differentiable w.r.t. sample positions to allow gradient flow from the reconstruction network (Subsection 3.3) to the importance network, so that the reconstruction network is trained simultaneously and propagates the loss information to the sampling stage. Since the entire pipeline is trained end-to-end using a loss on the reconstructed and ground truth images, the importance network and the pair of sampler and reconstruction network work together in an effort to learn the placement of samples so that high reconstruction quality is achieved.

In principle, one can refrain from using a separate importance map, by realizing the sampler as a network that directly learns the adaptive sampling. In this case, however, modeling the positional information in a network requires to represent positions explicitly, either in a graph structure or a linear field, so that less efficient graph networks or fully-connected networks need to be used. Furthermore, the sampler has to be re-trained whenever a different number of samples is used. Our approach enables the use of efficient convolutional networks, and to change the number of samples at testing time.

An overview of the processing pipeline is shown in Figure 2. It works with images comprised of an arbitrary number C of channels. In the first part of this work, the pipeline is introduced for isosurface rendering with $C = 5$, i.e., a binary mask (1: hit, 0: no hit), a normal vector, and a depth value. The application to direct volume rendered images is discussed in Section 6.

Weiss *et al.* [60] enforce frame-to-frame coherence during animations by including a temporal loss in the training step. This loss considers the difference between the previous frame – warped by the frame-to-frame optical flow – to the current frame. In the accompanying video, this approach is used for both the importance and reconstruction network. In the following discussion, however, temporal connections are omitted and the focus is solely on single image reconstruction for clarity.

3.1 Importance Network

The importance network $I \leftarrow \mathscr{N}_I(L)$ determines the distribution of the samples that are required by the reconstruction network to generate the visual output according to some loss function. Deeming every pixel equally important, i.e.,

$$\mathscr{N}_{I,\text{constant}}(L)_{ij} = \mathbf{1}, \tag{1}$$

leads to a uniform distribution of the samples [64, 21, 60]. Alternatively, and in the spirit of classical edge detection filters, the screen space gradients of the individual channels can be used, i.e.,

$$\mathscr{N}_{I,\text{gradient}}(L)_{ij} = \sum_c w_c ||\nabla L_{ij,c}||_2^2, \tag{2}$$

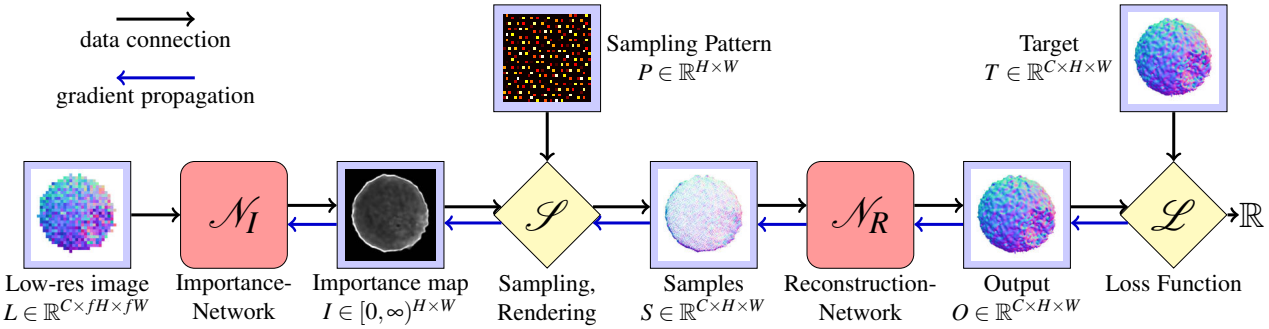


Fig. 2: Overview of network-based adaptive sampling. From a low-resolution image L, the importance network infers the importance map I. The sampler $\mathcal{S}$ uses this map together with a sampling pattern P to adaptively place samples in the high-resolution image S. Ray-casting the object at these samples generates a sparse image. The reconstruction network recovers the dense output O.

where $\nabla L_{ij,c}$ is the screen space gradient of channel c at location ij. The contributions of the individual channels are weighted by $w \in \mathbb{R}^C$. Other known importance measures consider screen space curvature via the variation of surface normals [46], or color contrast via the variation of luminance [58].

Alternatively, we introduce a fully convolutional neural network $\mathcal{N}_{I,\text{net}}$ (Subsection 4.2) that predicts a high-resolution greyscale importance map I from a low-resolution rendering L. Notably, this network is not trained w.r.t. specific characteristics that are derived from the image like gradients or luminance information, since this requires to heuristically decide on the importance of pixels. Instead, it is trained end-to-end with losses only on the reconstructed color information, by gradient descend all along the processing pipeline. In Section 5, network-based inference of the importance map is compared to alternative approaches, showing superior prediction of regions that are important for the final image.

3.2 Differentiable Sampling

Given the target number of samples in the final image, e.g. $\mu = 5\%$ of all pixels, the sampler uses the importance map I to determine where to place these samples. To generate the given number of samples, two main classes of algorithms are commonly used in rendering:

- Stippling starts with a given number of points at random locations and iteratively optimize these locations so that the point density matches the density of the importance map [10, 18].
- Importance sampling treats the importance map as a density function and place samples via rejection sampling or the inverse cumulative distribution function [35, 1].

These algorithms, however, are not easily differentiable w.r.t. changes in the importance map, since they use discrete optimizations or random processes, and often are too slow for real-time applications. To make the sampling process differentiable and fast, we propose a sampling strategy that computes for every pixel independently the chance of being sampled. This is achieved by a smooth approximation of rejection sampling, which is differentiable and allows for gradient propagation through the network pipeline. Since every pixel can be processed independently, this scheme can effectively leverage parallel execution on the GPU. On the other hand, it does not allow for an exact match of the prescribed number of samples, yet produces a number of samples that slightly varies around the target number.

In a first step, the importance map I is normalized to have a prescribed mean μ and minimal value $l \leq \mu$. Let μ_I be the mean of I over all pixels, then the image

$$I'_{ij} := \min\left\{1, l + I_{ij}\frac{\mu - l}{\mu_I + \varepsilon}\right\} \tag{3}$$

has the desired properties. A small constant $\varepsilon = 10^{-7}$ is used to avoid division by zero. The minimal value l is required to maintain a lower bound on the sample distribution in empty areas, which is important to allow for an accurate reconstruction in such areas. We use $l = 0.002$ in all of our experiments. Clamping to a maximal value of 1 is required by the following sampling step, which is realized as an independent Bernoulli process via rejection sampling, i.e., a sample at location ij is taken if the probability I'_{ij} is larger than a uniform random value $x \in [0,1]$.

To make the sampling deterministic and parallelizable on the GPU, a sampling pattern $P \in [0,1]^{H \times W}$ – uniformly distributed in $[0,1]$ – is first generated by using a permutation of the numbers $\frac{1}{HW}\{0,\ldots,HW-1\}$. We analyze four different strategies for generating the permutations: Random sampling, regular sampling, Halton sampling [19], and plastic sampling [50]. Plastic sampling has been selected since it produced slightly superior results in all of our experiments. Section B provides a detailed evaluation of the different strategies.

Ray-casting is then used to compute what is seen through the pixels at the determined sample locations. This information is stored in the high-resolution image $S \in \mathbb{R}^{C \times H \times W}$. Since during training the same view is rendered many times using different sampling patterns, pre-computed high-resolution target images $T \in \mathbb{R}^{C \times H \times W}$ are provided with the low-resolution inputs. Then, the sampling process simply becomes a selection of pixels from T:

$$S_{ij} = \mathbb{1}_{I'_{ij} - P_{ij}} T_{ij}, \tag{4}$$

where $\mathbb{1}_x$ is 1 if $x > 0$ else 0. Since the sampling function in Equation 4 is a step function with zero gradients almost everywhere, it is not differentiable w.r.t. the importance map I, from which I' is derived. Correspondingly, gradients in the loss function w.r.t. the weights and biases of the importance network will also be zero. Therefore, Equation 4 is approximated with a smooth sigmoid function to make it differentiable, so that gradients of the loss function can be back-propagated through all network stages to change the importance map accordingly. Then, the sampling function becomes

$$S_{ij} = \text{sig}\left(\alpha(I'_{ij} - P_{ij})\right) T_{ij}, \quad \text{sig}(x) := \frac{1}{1+e^{-x}}, \tag{5}$$

where $\alpha > 0$ determines the steepness of the function. The differentiable approximation is used only in the training phase, while in the validation phase the ray-caster renders the discrete samples obtained via rejection sampling. For $\alpha \to \infty$, Equation 5 converges to Equation 4. A large value of α leads to samples that are either very close to 0 or 1, but leads to exploding gradients in the backward pass. A low value leads to samples that smoothly cover the entire interval between 0 and 1. In this case, however, the mismatch between the "fractional" samples that are used only during training and the discrete "binary" samples that are used for testing and validation leads to a significant reduction of the reconstruction quality. In our experiments, a value of around $\alpha = 50$ always leads to the best results. Going beyond 50 quickly introduces floating-point precision issues and exploding gradients thereof. An evaluation of the dependency between the value of α and the reconstruction quality is provided in Subsection 5.2.

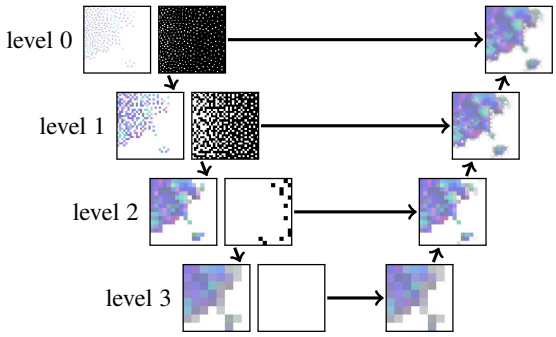


Fig. 3: Pull-push-based inpainting using a mipmap hierarchy of image samples and masks. The image is downsampled until all pixels are filled, and then upsampled by combining interpolated values from lower levels with the pixels at the current level. Masks are propagated through the hierarchy to obtain proper interpolation weights.

3.3 Differentiable Reconstruction

Given the sparse set of samples S, the reconstruction function $\mathcal{N}_R$ needs to estimate the undefined pixel values to produce the dense high-resolution output image $O \in \mathbb{R}^{C \times H \times W}$. By using a differentiable reconstruction function, gradients of the loss function on the reconstructed images and the ground truth image can be back-propagated through the sampling stage to the importance map.

In principle, there are different possibilities to fulfill the requirement of differentiability: Firstly, a neural inpainting network can be trained on sparse inputs and the ground truth outputs to learn the reconstruction. However, as we have verified in a number of experiments, network-based inpainting [25, 39, 62] at a sparsity level as used in our application leads to low reconstruction quality (see Figure 6b). The highly varying sample density with gaps between valid pixel values of up to 20 pixels poses a challenging problem for known network architectures. Furthermore, since during training the sampling mask in our proposed pipeline is not binary but contains continuous values, techniques like Partial Convolutions [39] are not applicable.

Secondly, classical non-network-based inpainting methods can be employed, for instance, PDE-based methods solving a constrained Laplace problem [4, 14], or patch-based methods using non-local cost functions involving correspondence functions [24, 12, 8]. These methods, however, are not easily differentiable w.r.t. the sampling mask. For example, PDE-based methods use the samples as Dirichlet boundaries and, to the best of our knowledge, there is no meaningful interpretation of a "fractional" Dirichlet boundary. Patch-based methods, on the other hand, use a discrete search over the image space to find a correspondence function, which makes the derivation of continuous gradients impossible.

Therefore, we introduce a novel reconstruction approach that combines a differentiable inpainting method with a residual neural network that learns to improve the inpainting result. In particular, we propose a variation of the pull-push algorithm [15, 32], which is differentiable with respect to the sampling mask and can cope with a mask that comprises fractional values.

The pull-push algorithm builds upon the idea of mipmap hierarchies. Firstly, the sparsely sampled high-resolution image and the mask are recursively filtered and downsampled by a factor of 2. The pixel values are averaged using the fractional values in the sampling mask as weights (average pooling), and max-pooling is used to combine the values in the mask. This has the effect of filling the undefined pixels with values that are averaged from a gradually increasing surrounding. Upon reaching a termination criterion, either a maximal number of steps or complete restoration of the undefined pixels, the images are bilinearly upscaled again. During upscaling, the pixel values from the coarse levels are weighted by the values in the mask at this level, and they are then blended with the value at the fine level based on the sampling values at that level. This allows to smoothly transition from filled pixels at the fine level that are kept in the output towards interpolated values for lower values in the sampling mask. A schematic illustration of the process is shown in Figure 3. Since the algorithm makes use exclusively of continuous pooling and interpolation operations, it is fully differentiable with respect to changes in the pixel data and the sampling mask. The forward code and a manually derived backward code are given in Section D. The algorithm has been implemented via custom CUDA operations in PyTorch [45].

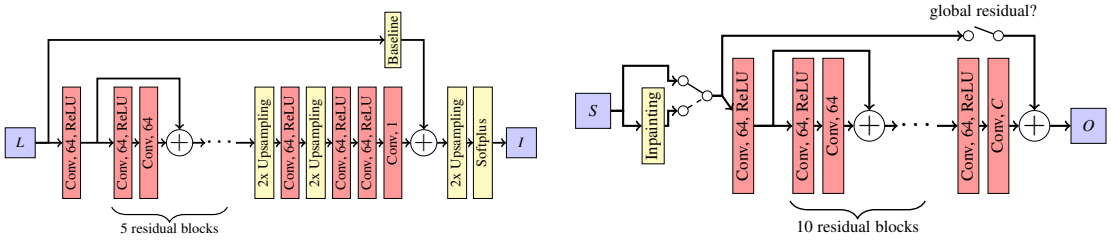
After inpainting the sparse samples via the pull-push algorithm, a fully convolutional network is used to improve the reconstruction by modeling the relationship between the inpainting result and the ground truth. The network sharpens the results and resolves blurred silhouettes created by the inpainting algorithm. We use the EnhanceNet [53] as base architecture for this learning task, which is discussed in detail in Subsection 4.2. In particular, we use the EnhanceNet as a residual network that starts with the inpainting result and learns to infer the changes to the reconstructed samples. A quantitative comparison of different learning approaches is provided in Subsection 5.2.

4 Training Methodology

In this chapter, we provide a detailed discussion of the used network architectures, as well as the training and inference steps. We also shed light on the dependency of the reconstruction quality on the used loss functions.

4.1 Training Data

As training and validation input, 5000 images of randomly selected isosurfaces in the Ejecta data set, a particle-based supernova simulation, were generated via GPU ray-casting at a screen resolution of 512^2. Each time step was resampled to Cartesian grids with a resolution of 256^3 and 512^3. The surfaces were rendered from random camera positions, at a varying distance to the object and always facing the object center. Renderings are



(a) Importance Network $\mathcal{N}_I$ (b) Reconstruction Network $\mathcal{N}_R$ (EnhanceNet)

Fig. 4: Network architectures used in the proposed pipeline: To estimate the importance map, we use a smaller version of the EnhanceNet [53] with a 4x-upsampling factor, a residual connection with screen space gradient magnitude as a baseline, and a 2x-upsampling network as a post-process. For the reconstruction network, we experimented with the option of passing the raw samples or interpolated samples as input and using a global residual connection or not. As network architecture, an EnhanceNet with 10 residual blocks is used.

taken from different time steps and resolution levels to let the pipeline learn features at different granularity [60]. The renderer provides the normals at the surface points, which are used in a post-process to compute colors via the Phong illumination model. From this image set, about 20.000 random crops of size 256^2 and showing the isosurface in at least 50% of the pixels were taken, and split between training (80%) and validation (20%). For training, the mean importance value was set to $\mu = 0.1$, i.e., 10% of the samples (see Equation 3). This does not prohibit using fewer samples for validation and testing, yet we found it beneficial to allow the network to use more samples during training. We used the Adam [30] optimizer with a learning rate of 10^{-4}. The networks were trained on a single GeForce GTX 1080 for 300 to 500 epochs in around 5-6 days.

4.2 Network Architectures

The proposed sampling pipeline comprises two trainable blocks: The importance network $\mathcal{N}_I$ and the reconstruction network $\mathcal{N}_R$. Both networks use 3x3 convolutions with zero-padding and a stride of one. The importance network is a variant of EnhanceNet [53], yet with only 5 residual blocks (Figure 4a). Instead of directly estimating the importance map, the network takes as input an importance map that is computed using screen space gradient magnitudes (Subsection 3.1), and learns to improve this map using a residual connection. We refer to Subsection 5.2 for a quantitative comparison of the network results w/ and w/o an initial gradient-based importance estimate.

The importance network performs 4x-upscaling of a low-resolution input image with $1/4$ the resolution of the final image. Thus, generating the input image requires to sample $1/4^2 = 6.25\%$ of the pixels in the target image, which already exceeds a prescribed limit of, e.g., 5% of the pixels. Therefore, an image with $1/8$ the final resolution is used as input, and the network performs 4x-upscaling to an intermediate image with $1/2$ the final resolution, followed by an additional 2x-upscaling of the inferred importance map. This allows to more aggressively reduce the number of initially required samples, i.e., only $1/8^2 \approx 1.56\%$ of the pixels in the final importance map need to be rendered.

The reconstruction network $\mathcal{N}_R$ estimates the mask, normal, and depth values at all pixels, thereby also changing the initial values that were drawn in the rendering process. A modified EnhanceNet (Figure 4b) shows superior reconstruction results compared to alternative architectures such as the U-Net [51]. Let us refer to Section A for a more detailed analysis of both architectures. Both networks are provided in the code repository accompanying this paper.

Our experiments (Subsection 5.2) show improved reconstruction quality if inpainting is performed first and the result is then passed to a network that uses a residual connection to learn the differences between this result and the ground truth. In addition to the inpainted input samples, we pass the sample mask to the network as a per-sample measure of certainty. Since the network produces output values in $\mathbb{R}$, both the mask and depth values are clamped to $[0,1]$, and the normals are scaled to unit length before shading is applied.

4.3 Loss Functions

We employ regular vector norms between the network prediction O and the target image T as primary loss functions on the individual output channels. Since the L_2 norm tends to smooth out the resulting images, we make use of the L_1 norm in this work. With the channels of the output image, i.e., the mask M, the normal map N, and the depth D, given as subscript, the L_1 loss of a selected channel X is

$$\mathscr{L}_{1,X} = ||T_X - O_X||_1. \tag{6}$$

We do not employ additional perceptual losses, which were shown less effective for isosurface upsampling tasks [60].

The mask channel has a special meaning as it indicates whether or not a ray hits the isosurface. It is used in the final output to perform a hard selection between the reconstructed color values and the background. To make the mask differentiable, however, its values must be continuous, leading to a smooth blend rather than a binary decision. While this is acceptable along the silhouettes, in the interior it would noticeably distort the reconstruction. In principle, via a sigmoidal mapping it can be enforced that the mask values spread continuously between 0 and 1, yet we observed undesirable blurring when using this approach. To produce sharp masks that are either close to zero or one, we therefore constrain the reconstruction via two losses that are added to the regular L_1 loss on the mask. The first loss is a binary cross entropy (BCE) loss that "pulls" the values closer to either zero or one than a normal L_1 loss:

$$\mathscr{L}_{\text{bce}} = -\frac{1}{WH}\sum_{ij}\left(T_{M,ij}\log(O_{M,ij}) + (1-T_{M,ij})\log(1-O_{M,ij})\right). \tag{7}$$

The BCE loss, however, requires that the output mask lies within $[0,1]$ and thus the mask is clamped beforehand. This leads to zero

gradients once the mask reaches values outside of $[0,1]$. Therefore, we add the loss

$$\mathscr{L}_{\text{bounds}} = \frac{1}{WH}\sum_{ij}\left(\max(0,(2O_{M,ij}-1)^2-1)\right), \tag{8}$$

which pushes values outside $[0,1]$ back into $[0,1]$ and leaves values within $[0,1]$ unchanged.

An additional loss term is required to account for the normalization step in Equation 3. The output of the importance map is normalized to limit the number of available samples. Hence, scaling the network output does not influence the values after normalization. Therefore, during training, the output values may increase or decrease in an unbounded manner. To prevent this, a prior on the importance map is used to enforce that the mean is equal to one before the normalization step:

$$\mathscr{L}_{I,\text{prior}} = \left(1-\frac{1}{WH}\sum_{ij} I_{ij}\right)^2. \tag{9}$$

The final loss function is a weighted sum of the individual loss terms over all channels, i.e., with $X \in \{M,N,D\}$ it becomes

$$\mathscr{L} = \sum_X \lambda_X \mathscr{L}_{1,X} + \lambda_{\text{bce}}\mathscr{L}_{\text{bce}} + \lambda_{\text{bounds}}\mathscr{L}_{\text{bounds}} + \rho\mathscr{L}_{I,\text{prior}}. \tag{10}$$

Loss weights around $\lambda_M = 5, \lambda_{\text{bce}} = 5, \lambda_{\text{bounds}} = 0.01, \lambda_N = 50, \lambda_D = 5$, and $\rho = 0.1$ lead to equally good reconstruction quality, while deviations from these values quickly worsen the reconstruction quality significantly.

5 Results and Evaluation

In the following, we evaluate the proposed network pipeline. First, we introduce the quality metrics that are used to compare the results. We then analyze how our design decisions influence the reconstruction quality on the validation data (Subsection 5.2). These statistics help to identify the network configurations with the best predictive skills. Next, the proposed network pipeline is compared to a fixed super-resolution network (Subsection 5.3). Finally, we shed light on the generalizability of the network pipeline to new views of Ejecta and data sets that were never seen during training (Subsection 5.4).

5.1 Quality Metrics

The quality of network-based reconstruction is assessed using three different image quality metrics commonly used in image processing. These metrics compare the output O of the network pipeline with a ground truth rendering T at the target resolution.

The peak signal-to-noise ratio (PSNR) is based on the L_2 loss and is defined as

$$\text{PSNR}(O,T) = -10\log_{10}(||O-T||_2^2), \tag{11}$$

where O and T are the network output and target image, respectively.

The Structural Similarity Index (SSIM) [59] extends on the idea of per-pixel losses by measuring the perceived quality using the mean and variance of contiguous pixel blocks in the images. It is defined as

$$\text{SSIM}(O,T) = \frac{(2\mu_O\mu_T + c_1)(2\sigma_{O,T} + c_2)}{(\mu_O^2+\mu_T^2+c_1)(\sigma_O^2+\sigma_T^2+c_2)}, \tag{12}$$

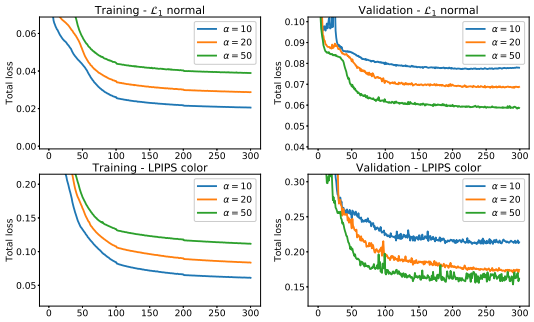


Fig. 5: Influence of the sharpness parameter α on the training process. A lower value leads to a lower cost during training, but increases the cost in the validation phase where a perfect step function is used.

where μ_O and μ_T are the average values of O and T, σ_O^2 and σ_T^2 are the variances of O and T, $\sigma_{O,T}$ is the covariance between O and T, and c_1 and c_2 are small constants to avoid division by zero.

We also use the network-based Learned Perceptual Image Patch Similarity (LPIPS) metric [63] that predicts human perception of relative image similarities. LPIPS builds upon a network that is pre-trained on an image classification task—using the AlexNet [33]—and computes a weighted average of the activations at hidden layers for a given output and target image. Note that a lower LPIPS score is better, whereas PSNR and SSIM indicate higher quality by a higher score. Therefore, $1-\text{LPIPS}$ is shown in our statistics for better comparison.

5.2 Validation of Design Decisions

Unless otherwise mentioned, all statistics presented in this section were computed on a validation data set using 2000 novel views of Ejecta at the resolution of 512^2. The importance map was normalized to have a minimal value $l = 0.002$ and a mean value $\mu = 0.05$. "plastic" sampling was used in the sampling stage.

Steepness of the Sampling Function The parameter α in Equation 5 determines the steepness of the sampling function. A perfect step function, as used for testing, is obtained for $\alpha \to \infty$. Figure 5 compares the total loss on the training and validation data over the course of the optimization for different values of α . A lower value of α leads to a lower cost on the training data, because smoother variations in the fractional samples can be used for reconstruction. However, this behaviour is reversed during validation, because the perfect step function corresponds to a lesser and lesser extent with an increasingly smooth sampling function. Higher values of α, on the other hand, lead to better generalization, yet beyond 100 we observed instabilities in the training as well as numerical precision issues. Therefore, we decided to use $\alpha = 50$ in all of our experiments.

Residual Connections for Reconstruction In principle, there are different options to reconstruct a dense image from the sparse set of samples, including sole inpainting via the pull-push algorithm as well as inpainting in combination with network-based reconstruction w/ or w/o residual connections. In Figure 6, the reconstruction quality of all options is compared, using screen space gradient magnitudes as measure for generating the importance map.

As can be seen, the pull-push algorithm already provides a good initial guess on the reconstructed image, and reconstruction quality reduces significantly when it is not used. On the other hand, the network-based approach fails to reliably fill the empty pixels, which is probably due to the vastly different distances between the

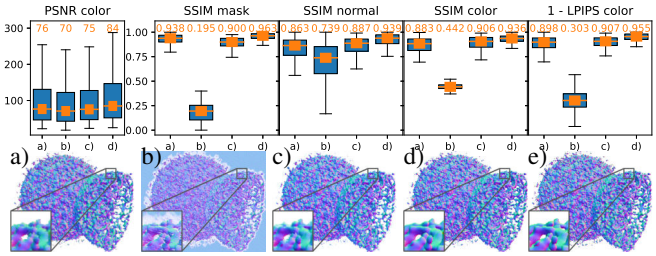


Fig. 6: Comparison of different reconstruction methods: (a) Only pull-push-based inpainting. (b) Only network-based reconstruction without residual. (c) Pull-push plus reconstruction network w/o residual. (d) Pull-push inpainting plus reconstruction network with residual. (e) Ground truth. An importance map from screen space gradient magnitudes is used in all examples, with $\mu = 5\%$ of samples.

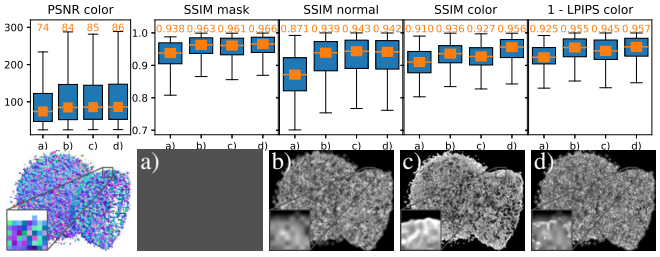


Fig. 7: Reconstruction quality using different importance maps. Bottom left: Low-resolution input. (a) Constant map, (b) based on gradient magnitudes, (c) only network-based learning, (d) network-based learning with residual on gradient magnitudes. $\mu = 5\%$ of samples were used. Top: Quality metrics for options (a) to (d).

sparse samples. When using the pull-push algorithm in combination with network-based reconstruction, but with disabled residual connections, no benefit over sole pull-push-based inpainting is gained. The best result is achieved with both pull-push-based inpainting and residual network connections. This is in line with the findings of Kim *et al.* [28], that the quality of network-based reconstruction improves if the network needs to learn only the changes to the baseline method.

Residual Connections for Importance Mapping On the validation data, we then analyze the reconstruction quality using different approaches for generating the importance map, i.e., constant importance, importance derived from screen space gradient magnitudes, as well as network-based importance with or without learning a residual to screen space gradient magnitudes. Figure 7 shows the results using the quality metrics described above.

As expected, screen space gradient magnitudes already hint at some important regions that should be sampled with higher density, significantly outperforming a constant importance map. For reconstructing the mask and normal channels, gradient magnitudes and network-based importance learning differ only marginally w.r.t. reconstruction quality. The importance network puts more emphasis on the object silhouettes and leads to an improved reconstruction of the normals over gradient magnitudes. On the other hand, it is important to note that the network learns the importance of features for an accurate screen space reconstruction without any prior information (Figure 7c). The best results are achieved by combining network-based importance learning and screen space gradient magnitudes via a residual network connection, demonstrating the feasibility of learning features that are important for an accurate reconstruction.

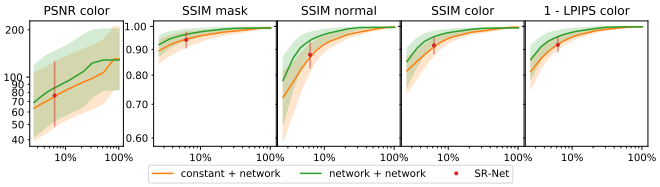


Fig. 8: Median reconstruction quality and 25% / 75% quantiles shown as confidence bands for increasing number of samples. Orange: Network-based pipeline using a constant importance map. Green: Network-based pipeline with the network-based importance map. The red dot represents the 4x-upsampling network from Weiss *et al.* [60].

5.3 Convergence and Regular Sampling

We further analyze the convergence of the proposed sampling pipeline with an increasing number of samples. The network is trained with 10% of the samples, but during inference the available number of samples is varied. The results in Figure 8 indicate, that with an increasing number of samples the SSIM and LPIPS scores converge against their optima. Even though this seems logical at first, since the reconstruction network modifies the given samples, it could, in principle, converge against some other solution. Notably, already after taking 20% to 30% of samples, the reconstruction is very close to the target.

We also compare the quality of adaptive sampling to fixed regular sampling using a 4x-upsampling network [60]. The 4x-upsampling network uses a regular sampling structure comprised of $1/4^2 = 6.25\%$ of the pixels in the high-resolution image, corresponding to a constant importance map with 6.25% of the samples when adaptive sampling is used. Figure 8 shows that the 4x-upsampling network (red) performs equally good as the adaptive pipeline using a constant importance map (orange). However, when the samples are placed adaptively according to the inferred importance map (green), the reconstruction quality is significantly increased at the same number of samples.

5.4 Generalizability

The importance and reconstruction networks are trained solely on Ejecta. To test how well the networks generalize, they are applied to several data sets that were never seen during training. We use a Richtmyer-Meshkov (RM) simulation at $1024 \times 1024 \times 960$, CT scans of a human skull (Skull) at 256^3, an aneurism (Aneurism) at 256^3, a bug (Bug) at $416^2 \times 247$, and a human body (Thorax) at $256^2 \times 942$, as well as a jet stream simulation (Jet) at 256^3. Quantitative statistics for RM, Skull and novel views of Ejecta are given in Figure 9. Reconstructed images as well as SSIM and LPIPS statistics for all data sets are shown in Figure 10.

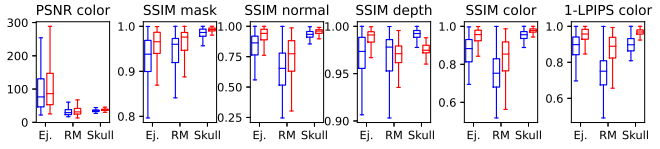


Fig. 9: Quality statistics for novel views of Ejecta and new data sets RM and Skull (see Figure 10). Baseline method (blue) refers to gradient magnitude-based importance mapping and pull-push-based inpainting. Results of the proposed network pipeline are shown in red.

The pipeline generalizes well to new data sets and views, and it performs better than the baseline method using gradient magnitude-based importance mapping and pull-push-based inpainting. In

particular, the network pipeline produces a tighter spread of the quantitative measures in general, indicating less significant outliers in the reconstructed values. The network shows lower scores only for the depth maps reconstructed from sparse samples of RM and Skull. We attribute this to different zoom levels in the renderings and the training images, yet these inaccuracies do not affect the quality of the reconstructed color images. For reconstruction, we also analyzed the quality of other inpainting algorithms such as PDE-based methods. Notably, these methods are not differentiable and, thus, cannot be used for end-to-end training in combination with the importance network, yet they can be used for sole sparse image reconstruction. A comparison to the pull-push algorithm, however, does not show any perceivable differences. The results further indicate that the network pipeline can reconstruct images at high fidelity from only 5% of the samples that are used to render the data sets at full pixel resolution. In particular sharp edges are well preserved, since the network has learned to increase the sample density along them.

6 Application to DVR

The proposed network pipeline can be applied to images that are rendered via Direct Volume Rendering (DVR), i.e., volume ray-casting using an emission-absorption model along the rays of sight. In contrast to isosurface ray-casting, not only one single ray-surface intersection point is rendered, but the colors of many sample points along the rays are blended using α-compositing to account for volumetric attenuation.

6.1 Training and Validation

The importance and reconstruction networks receive RGBα images as input, and the network pipeline outputs the reconstructed high-resolution RGBα images. Interestingly, we observed a noticeable increase in quality when the gradients at the sample points along the view rays are used by the importance and reconstruction network. The normalized gradients in $[-1,1]^3$ along a single ray are treated as emission and blended according to the volume rendering integral, just as blending the RGB colors. The resulting gradient map is then used as an additional input channel. Since the average gradients indicate, to a certain extent, whether two rays step through vastly different or similar regions, the gradient map serves as an additional coherence indicator. When only a single isosurface is rendered, the resulting values converge against the values in the normal map.

For training and validation, random transfer functions (TFs) are generated and used to render Ejecta, with L_1 losses on color and alpha in combination with an LPIPS-based perceptual loss (Section C). Since the low-resolution input to the importance network is also generated with a TF, the network can learn to select features specific to that TF, even though this was never seen during training. It is important to note that the reconstruction quality strongly depends on the use of TFs that include a broad range of different colors in the training step. For instance, if the training data only contains desaturated colors, strongly saturated colors during testing cannot be reconstructed.

6.2 DVR Results

For novel views of Ejecta and the data sets introduced in Subsection 5.4, Figure 11 shows a qualitative analysis of the results of importance sampling and reconstruction using DVR as well as SSIM and LPIPS statistics. None of these data sets was used in the training and validation phases, and the results have been generated using TFs that were never seen during training. The results indicate that the network pipeline generalizes well to new volumes and TFs, yet the reconstruction quality is affected by the occurring color variations. Especially for Thorax and Aneurism, where the TFs introduce rather small-scale color variations in some areas, in these areas the network places the samples rather uniformly and, thus, cannot accurately reconstruct the rendered structure. Overall, it can be seen that the reconstruction problem is significantly more challenging when using DVR samples instead of isosurface samples. When rendering isosurfaces, the shading in the interior of the rendered structures is rather smooth, enabling the network to focus on the silhouettes and internal edges. In DVR, on the other hand, the network needs to learn both the shape and the color texture stemming from the application of a TF.

TABLE 1: Timings (in milliseconds) of network-based volume rendering (averaged over 100 different views at 1024^2 target resolution) for data sets shown in Figure 10 and Figure 11. Timings are for rendering the low-resolution input image (128^2) and the sparse set of samples (5% and 10% of the target resolution for isosurface rendering and DVR respectively), generating the importance map and sampling pattern, reconstructing the image, and GPU ray-casting at the target resolution.

	Test case	Rendering	Importance	Reconstruction	Total	GT
ISO	Ejecta 512^3	24.3	7.4	92.1	123.9	105.8
ISO	RM 1024^3	34.3	5.8	92.0	132.1	89.4
ISO	Skull 256^3	6.1	5.9	93.7	105.6	27.3
DVR	Ejecta 512^3	46.5	5.8	92.2	144.5	224.8
DVR	RM 1024^3	51.7	5.8	91.8	149.3	158.3
DVR	Thorax 512^3	15.9	5.9	92.9	114.7	63.7

7 Performance Analysis

Even though performance improvements are not our main objective, it is interesting to see whether network-based adaptive sampling and image reconstruction can be faster than full-resolution GPU ray-casting, due to the reduced number of samples that need to be taken. The following performance tests were carried out on a workstation running Windows 10 with an Intel Xeon E5-1630 @3.70GHz CPU, 32GB RAM, and an NVidia Titan RTX. All timings are averages over 100 frames with random camera positions, with the screen resolution set to 1024^2. The ray-caster uses a constant step size of 0.25 voxels and tricubic interpolation.

For some of the data sets shown in Figure 10 and Figure 11, Table 1 lists the times that are required by the pipeline stages and full-resolution volume ray-casting. Only for the larger data sets and DVR can the network pipeline achieve a slightly better performance than the ray-caster. Especially the reconstruction network consumes a significant portion of the overall time, sometimes even more than it requires to render at full resolution. This is because the reconstruction network requires a large amount of data access and arithmetic operations on the GPU, independent of the volume resolution.

On the one hand, the performance of the reconstruction network scales linearly with the number of pixels, and hence quadratically with the screen resolution. Volume rendering, on the other hand, scales quadratically with the screen resolution but also linear in the volume resolution. The sampling stage, even though it also scales in the volume resolution, performs a significantly smaller number of sampling operations than the full-resolution ray-caster. Thus, its overall contribution is negligible, so that performance benefits can be expected with increasing image and volume size. This is demonstrated in Table 2, where versions of RM and Ejecta at 2048^3

TABLE 2: Performance scaling w.r.t. image and volume size. Each entry shows the total time of the network pipeline (low-resolution rendering, importance network, sparse sampling, reconstruction network) and the time required by the volume ray-caster at full resolution. All timings are in milliseconds. The cells are colored with a diverging color map, encoding the performance differences from red (superior performance of ray-casting) to blue (superior performance of the network pipeline).

		Volume resolution	Screen resolution 256	512	1024	2048
ISO	Ejecta	256	24/11	45/20	115/75	398/294
		512	31/16	55/31	124/106	405/400
		1024	45/47	73/104	141/221	445/655
		2048	85/278	132/806	238/1358	724/2324
	RM	256	24/4	41/11	111/42	393/173
		512	31/6	51/15	116/58	397/236
		1024	45/15	71/31	132/89	411/366
		2048	87/122	124/249	211/453	646/802
DVR	Ejecta	256	46/17	64/46	131/162	413/574
		512	59/36	75/73	144/225	447/785
		1024	86/151	103/262	180/433	594/1274
		2048	160/575	186/1692	333/2771	1319/4292
	RM	256	32/8	49/21	117/71	398/280
		512	41/11	58/27	125/96	404/363
		1024	60/29	81/58	149/158	450/549
		2048	115/203	143/410	248/656	830/1304

are rendered at different resolution levels and large images sizes. Note that in these experiments an Nvidia Titan RTX graphics card with 24GB of memory was used to keep all data in memory.

It can be seen that for large image sizes—where the GPU is fully utilized by the network—and volume sizes larger than 1024^3, the network pipeline outperforms the GPU ray-caster. Even though a ray-caster using advanced acceleration schemes can achieve improved performance, we are confident that in these scenarios faster deep-learning hardware and performance-optimized network architectures will let the performance differences grow due to better scalability of the network pipeline.

8 Conclusion and Future Work

In this paper we have introduced and analyzed a network pipeline that learns adaptive screen space sampling and reconstruction for 3D visualization, with the focus on volume rendering applications. For the first time, to our best knowledge, a fully differentiable adaptive sampling pipeline comprised of an importance network, a sampling stage, and a reconstruction network is proposed. Our experiments have shown, that the pipeline learns to determine the locations that are important for an accurate image reconstruction, and achieves high reconstruction quality for a sparse set of samples.

We are particular intrigued about the quality of the results compared to sampling methods that consider explicitly certain feature descriptors. Even without such supervision, the network pipeline can improve on the reconstruction quality, using solely image-based quality losses. We believe that especially for data visualization there is value in the observation that artificial neural networks can learn the relevance of structures for generating visual representations. For sole rendering tasks, on the other hand, superior performance compared to classical volume ray-casting can only be achieved for large image and volume sizes.

The application to DVR opens the interesting question of whether the proposed network pipeline can be used beyond adaptive sampling in screen space, and learn where to sample in object space so that the relevant information is conveyed visually. Conceptually this requires end-to-end learning of a mapping from a low-resolution object space representation to a high-resolution visual representation. The ultimate goal is to let the network learn to convert a low-resolution input volume to a compact yet feature-preserving latent-space representation from which a highly accurate view can be inferred.

In particular, we envision a neural volume rendering pipeline, where during training a neural scene representation is built and trained end-to-end with a renderer that learns sampling and color mapping simultaneously. In the future, we will analyze whether a network can learn a suitable color mapping for a given volumetric field. We also see challenging research problems in the area of transfer learning, to infer the most important samples for training, and to generate synthetic volumetric fields to enable training in domains where training data is rare.

References

[1] T. Bashford-Rogers, K. Debattista, and A. Chalmers. Importance driven environment map sampling. *IEEE transactions on visualization and computer graphics*, 20, 11 2013. doi: 10.1109/TVCG.2013.258

[2] S. Belyaev, P. Smirnov, V. Shubnikov, and N. Smirnova. Adaptive algorithm for accelerating direct isosurface rendering on gpu. *Journal of Electronic Science and Technology*, 16:222–231, 01 2018. doi: 10.11989/JEST.1674-862X.71013102

[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, April 2019. doi: 10.1109/TVCG.2018.2816059

[4] M. Bertalmio, A. L. Bertozzi, and G. Sapiro. Navier-stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, Dec 2001. doi: 10.1109/CVPR.2001.990497

[5] 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

[6] L. Campagnolo, W. Celes, and L. Figueiredo. Accurate volume rendering based on adaptive numerical integration. pp. 17–24, 08 2015. doi: 10.1109/SIBGRAPI.2015.27

[7] M. Chu, Y. Xie, L. Leal-Taixé, and N. Thuerey. Temporally coherent gans for video super-resolution (tecogan). *arXiv:1811.09393*, 2018.

[8] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, Sep. 2004. doi: 10.1109/TIP.2004.833105

[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] O. Deussen, M. Spicker, and Q. Zheng. Weighted Linde-Buzo-Gray stippling. *ACM Trans. Graph.*, 36(6), Nov. 2017. doi: 10.1145/3130800.3130819

[11] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pp. 184–199. Springer, 2014.

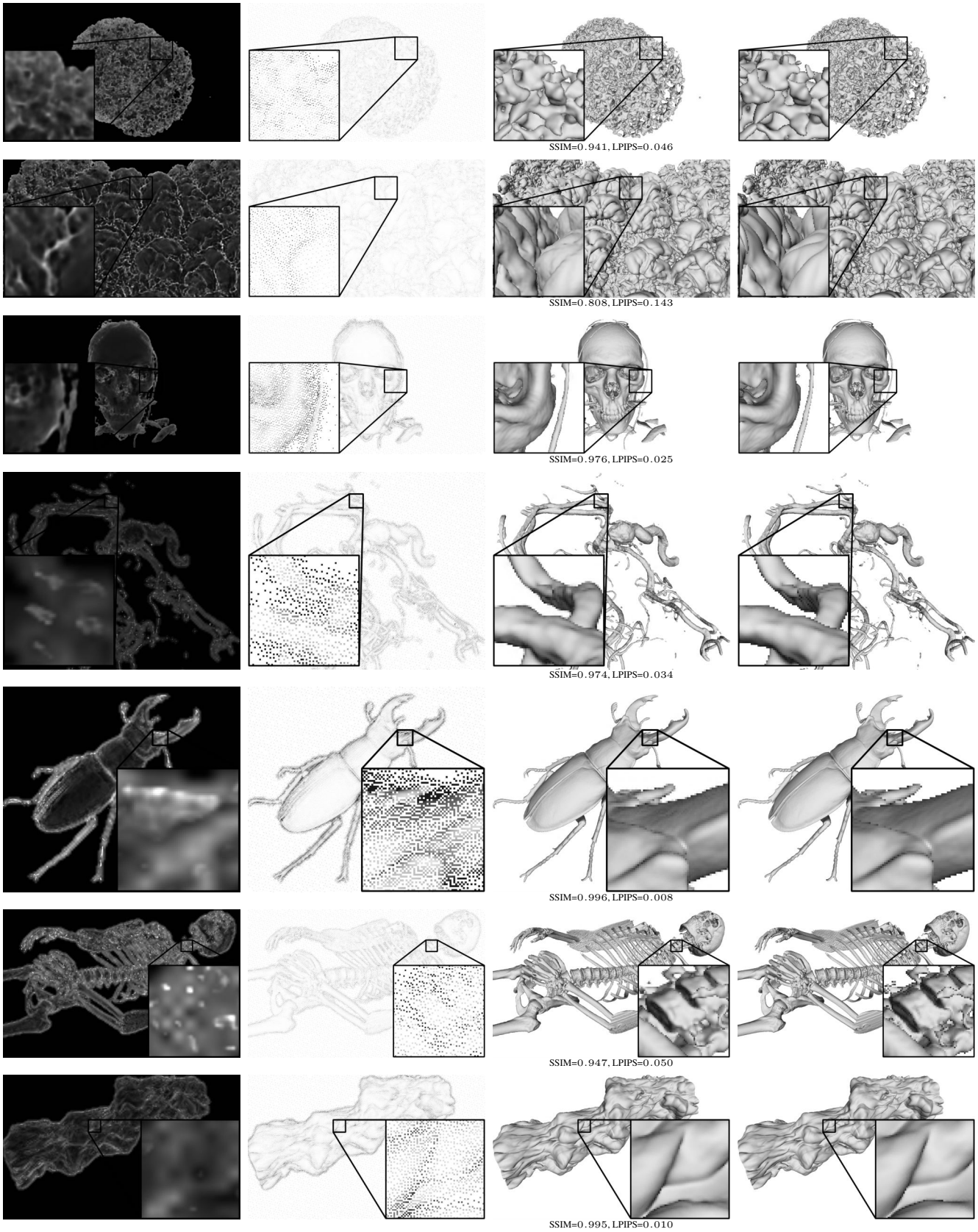


Fig. 10: Visual comparison of adaptive sampling of isosurfaces. From left to right: The importance map and, the sparse set of samples, the inpainted samples, the network output, the ground truth (normals and colors using reconstructed normals). From top to bottom: Novel views of Ejecta, RM, Skull, Aneurism, Bug, Human, Jet. Networks were trained only on Ejecta. The number of samples is $\mu = 5\%$ of the pixels in the output image.

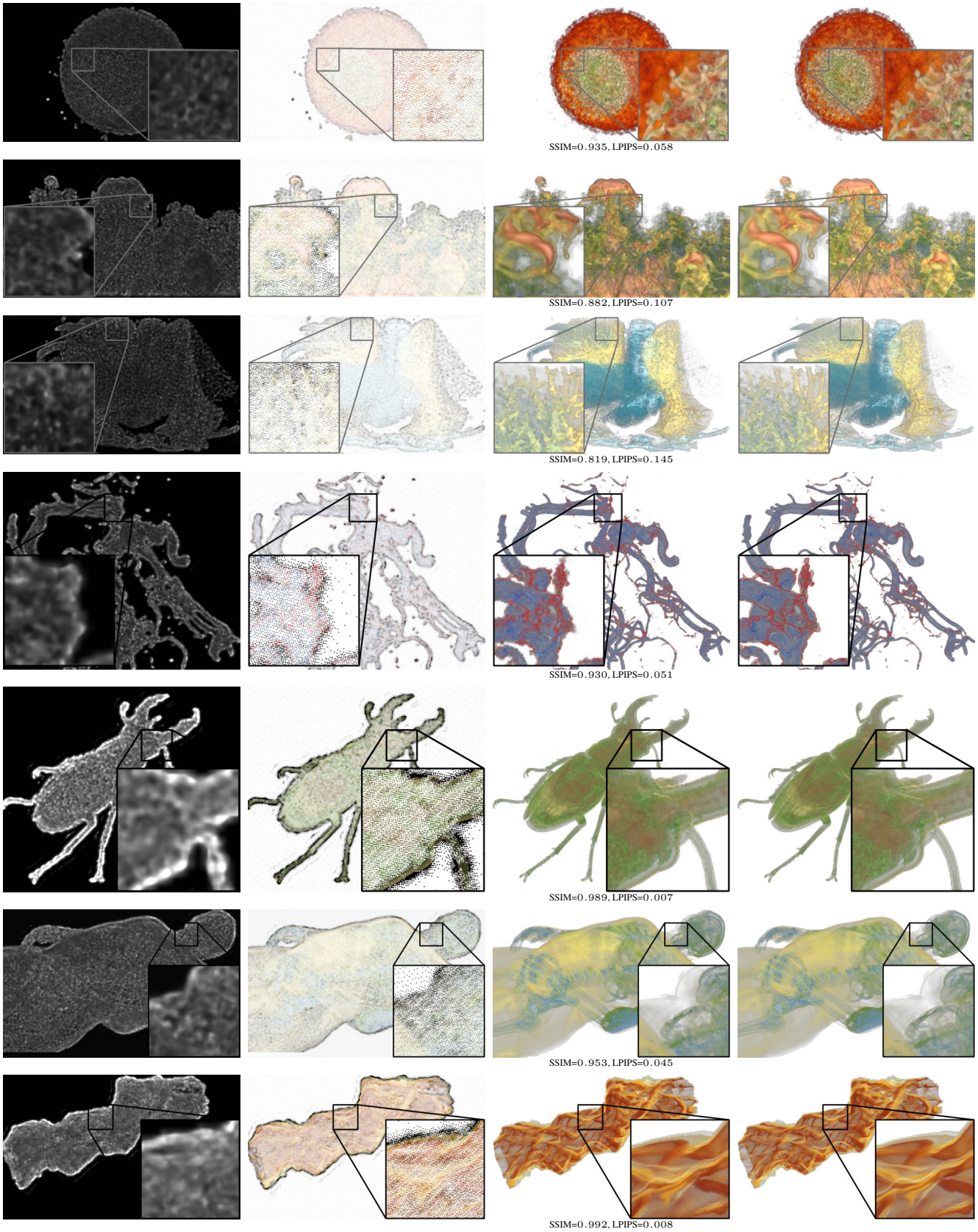


Fig. 11: Visual comparison of adaptive sampling for DVR. Each row shows – from left to right – the importance map, the sparse set of samples, the network output, the ground truth. From top to bottom: Novel views of Ejecta, RM, Thorax, Aneurism, Bug, Human, Jet. Networks were trained only on Ejecta. The number of samples is $\mu = 10\%$ of the pixels in the output image.

[12] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1033–1038 vol.2, Sep. 1999. doi: 10.1109/ICCV.1999.790383

[13] S. Frey, F. Sadlo, K. Ma, and T. Ertl. Interactive progressive visualization with space-time error control. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2397–2406, 2014.

[14] P. Getreuer. Total variation inpainting using Split Bregman. *Image Processing On Line*, 2:147–157, 2012. doi: 10.5201/ipol.2012.g-tvi

[15] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 43–54, 1996.

[16] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. Foveated 3d graphics. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

[17] L. Guo, S. Ye, J. Han, H. Zheng, H. Gao, D. Chen, J.-X. Wang, and C. Wang. Spatial super-resolution for vector field data analysis and visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, 2020.

[18] J. Görtler, M. Spicker, C. Schulz, D. Weiskopf, and O. Deussen. Stippling of 2d scalar fields. *IEEE Transactions on Visualization and Computer Graphics*, 25(6):2193–2204, June 2019. doi: 10.1109/TVCG.2019.2903945

[19] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, Dec. 1964. doi: 10.1145/355588.365104

[20] J. Han, J. Tao, H. Zheng, H. Guo, D. Z. Chen, and C. Wang. Flow field reduction via reconstructing vector data from 3-d streamlines using deep learning. *IEEE computer graphics and applications*, 39(4):54–67, 2019.

[21] J. Han and C. Wang. TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):205–215, 2019.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[23] W. He, J. Wang, H. Guo, K. Wang, H. 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, Jan 2020. doi: 10.1109/TVCG.2019.2934312

[24] H. Igehy and L. Pereira. Image replacement through texture synthesis. In *Proceedings of International Conference on Image Processing*, vol. 3, pp. 186–189 vol.3, Oct 1997. doi: 10.1109/ICIP.1997.632049

[25] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017.

[26] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pp. 694–711. Springer, 2016.

[27] J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016.

[28] J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[29] J. Kim, J. Kwon Lee, and K. Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1637–1645, 2016.

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

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

[32] M. Kraus. The pull-push algorithm revisited. *Proceedings GRAPP*, 2:3, 2009.

[33] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*, 2014.

[34] A. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi. Deep adaptive sampling for low sample count rendering. In *Computer Graphics Forum*, vol. 37, pp. 35–44. Wiley Online Library, 2018.

[35] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi. Adaptive numerical cumulative distribution functions for efficient importance sampling. In *Rendering Techniques*, pp. 11–20, 2005.

[36] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[37] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990.

[38] S. Lindholm, D. Jönsson, H. Knutsson, and A. Ynnerman. Towards data centric sampling for volume rendering. In *SIGRAD*, 2013.

[39] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[40] P. Longhurst, K. Debattista, and A. Chalmers. A GPU based saliency map for high-fidelity selective rendering. In *Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, AFRIGRAPH '06, p. 21–29. Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1108590.1108595

[41] M. Mara, M. McGuire, B. Bitterli, and W. Jarosz. An efficient denoising algorithm for global illumination. In *Proceedings of High Performance Graphics*. ACM, New York, NY, USA, jul 2017. doi: 10.1145/3105762.3105774

[42] K. Myszkowski. The visible differences predictor: applications to global illumination problems. In G. Drettakis and N. Max, eds., *Rendering Techniques '98*, pp. 223–236. Springer Vienna, Vienna, 1998.

[43] K. Novins and J. Arvo. Controlled precision volume integration. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS '92, p. 83–89. Association for Computing Machinery, New York, NY, USA, 1992. doi: 10.1145/147130.147154

[44] 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.

[45] 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.

[46] M. Prantl, L. Vása, and I. Kolingerová. Fast screen space curvature estimation on gpu. In *VISIGRAPP (1: GRAPP)*, pp. 151–160, 2016.

[47] C. R. Alla Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics*, 36:1–12, 07 2017. doi: 10.1145/3072959.3073601

[48] M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. A perceptually based physical error metric for realistic image synthesis. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, p. 73–82. ACM Press/Addison-Wesley Publishing Co., USA, 1999. doi: 10.1145/311535.311543

[49] J. Rigau, M. Feixas, and M. Sbert. Refinement criteria based on f-divergences. In *Rendering Techniques*, pp. 260–269, 2003.

[50] M. Roberts. The unreasonable effectiveness of quasirandom sequences. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/, 2020. Accessed: 2020-02-14.

[51] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241. Springer International Publishing, Cham, 2015.

[52] M. S. Sajjadi, R. Vemulapalli, and M. Brown. Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6626–6634, 2018.

[53] M. S. M. Sajjadi, B. Scholkopf, and M. Hirsch. EnhanceNet: Single image super-resolution through automated texture synthesis. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[54] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

[55] Y. Tai, J. Yang, and X. Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3147–3155, 2017.

[56] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia. Detail-revealing deep video super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4472–4480, 2017.

[57] G. Tkachev, S. Frey, and T. Ertl. Local prediction models for spatiotemporal volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019. doi: 10.1109/TVCG.2019.2961893

[58] O. T. Tursun, E. Arabadzhiyska-Koleva, M. Wernikowski, R. Mantiuk, H.-P. Seidel, K. Myszkowski, and P. Didyk. Luminance-contrast-aware foveated rendering. *ACM Trans. Graph.*, 38(4), July 2019. doi: 10.1145/3306346.3322985

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

[60] 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

[61] 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.

[62] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4471–4480, 2019.

[63] 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.

[64] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proceedings of the Computer Graphics International Conference*, pp. 1–6, 2017.

Sebastian Weiss received the M.Sc. degree from the Technical University of Munich in 2018. Currently, he is a doctoral student of computer science, Technical University of Munich. His research interests include volume visualization, deep learning and high performance GPU programming.

Mustafa Işık received his B.Sc. degree in Computer Engineering from Middle East Technical University, Turkey. Currently, he is pursuing an M.Sc. in Computer Science at Technical University of Munich, Germany. His research interests include realistic image synthesis, deep learning and image denoising.

Justus Thies is a Postdoctoral Researcher in the Visual Computing Group at the Technical University of Munich (TUM). He received his PhD from the University of Erlangen-Nuremberg in 2017 for his research on marker-less motion capturing of facial performances and its applications. More recently, he focuses on neural image synthesis techniques that allow for video editing and creation.

Rüdiger Westermann studied computer science at the Technical University Darmstadt and received his Ph.D. in computer science from the University of Dortmund, both in Germany. In 2002, he was appointed the chair of Computer Graphics and Visualization at TUM. His research interests include scalable data visualization and simulation algorithms, GPU computing, real-time rendering of large data, and uncertainty visualization.

APPENDIX A
COMPARISON WITH THE U-NET FOR RECONSTRUCTION

For reconstruction, we also tested different variants (by varying the number of levels and channels at each level) of the U-Net architecture [7]. As one can see in Figure 13, in our application the EnhanceNet vastly outperforms all considered U-Net variants.

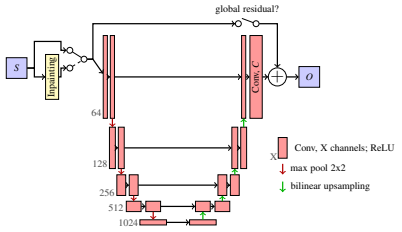


Fig. 12: Reconstruction network based on the U-Net architecture. See 4b for the EnhanceNet architecture we use in our experiments.

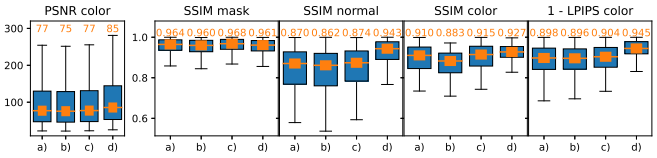


Fig. 13: Comparison of the U-Net and EnhanceNet for sparse image reconstruction: U-Net 4-4 (a), U-Net 5-3 (b), U-Net 5-4 (c), EnhanceNet (d). a-b indicates a levels and 2^{b+i} channels in level i (zero-based). The importance network is trained together with the reconstruction network.

APPENDIX B
COMPARISON OF DIFFERENT SAMPLING PATTERN

For deterministic and parallelizable sampling on the GPU, we use a pre-computed sampling pattern in combination with rejection sampling (Subsection 3.2). The sampling pattern $P \in [0,1]^{H\times W}$ contains permutations of uniformly distributed numbers in $[0,1]$, $\frac{1}{HW}\{0,\ldots,HW-1\}$. Here we analyze the four different strategies employed for generating the permutations (Figure 14, top): Random sampling, regular sampling, Halton sampling [2], and plastic sampling [6].

Random sampling generates a random permutation of the numbers in P. Regular sampling arranges the pixels in a quad-tree and enumerates them using breath-first traversal to generate the sampling pattern. Random and regular sampling introduce, respectively, largely varying sample densities and a strong bias of the sample distribution towards the top of the image. Both Halton and plastic sampling are deterministic and produce quasi-random sequences with a fairly uniform distribution. As revealed by the quantitative analysis in Figure 15, even though all sampling strategies allow reconstructing the final image at high accuracy, slight differences are noticeable. Halton and plastic sampling lead to superior quality, in particular w.r.t. the variance of the quality metrics. Plastic sampling, designed as a low-discrepancy sampling sequence, shows the lowest variance and slightly higher scores than Halton sampling. We, therefore, use plastic sampling in our implementation.

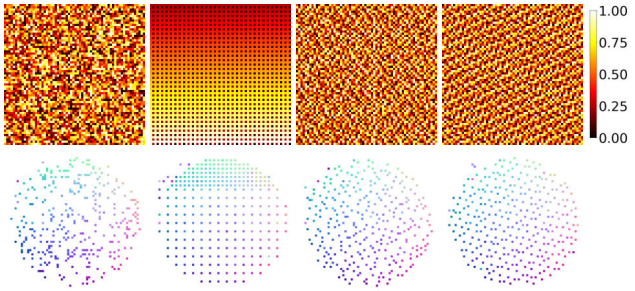


Fig. 14: Comparison of random sampling, regular sampling, Halton sampling and plastic sampling (left to right). Top: The sampling sequences. Bottom: The sequences applied to render a sphere with constant importance of $\mu = 0.1$ (shown are color coded normals at rendered fragments).

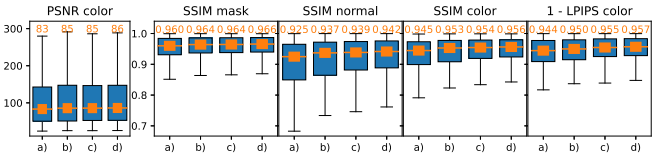


Fig. 15: Reconstruction quality using (a) uniform random, (b) regular, (c) Halton (c), and (c) plastic sampling with $\mu = 5\%$ of samples.

APPENDIX C
APPLICATION TO DVR

In this section, we provide additional details on how the proposed adaptive sampling pipeline is applied to DVR images, as mentioned in Section 6. First, we present the changes to the pipeline in terms of input and output channels and the used loss function. Second, we describe how to generate the training data including the sampling of transfer functions.

Input Channels and Loss Function: First, the input channels to the network pipeline are reinterpreted. For isosurfaces, a mask, normals, and depth were passed to the network as input (5 channels per pixel), now color images from the DVR images, together with alpha, depth, and normal maps are used as input (8 channels per pixel). The network also only reconstructs color images in RGBα space.

For DVR, depth and normal maps are computed by treating the screen space depth and normal at each sample in object space as a regular color and blended as such with the opacity given by the transfer function (TF). The result is a single depth and normal value per ray which can be interpreted as a weighted average of the depth and normal of all samples along the ray. We found that adding depth and normals as input channels improves the quality of the reconstruction as it provides additional locally consistent information about the curvature of the object.

Second, the loss functions on the individual channels as used for isosurfaces are replaced by losses only on the RGBα-color. We apply L_1 losses on the color and alpha and an additional LPIPS metric [10] as a perceptual loss, weighted equally:

$$\mathscr{L}_{\text{dvr}} = \mathscr{L}_{1,rgba} + \mathscr{L}_{\text{LPIPS},rgb}. \tag{13}$$

We found that adding a perceptual loss is critical in reconstructing fine details and sharp silhouettes. The networks operate in RGB space, other colorspaces like HSV, XYZ, or CIELAB did not improve the result. Furthermore, the training data is augmented by randomly shuffling the RGB channels. This helps the network to not overfit for a specific color.

Data Set Generation: For training and validation, random transfer functions (TFs) are generated (see below) and Ejecta was used as data set. The test images in the result section use user-generated TFs. Note that since the low-resolution input for the importance network is also generated with a TF, the network can learn to select features specific to that TF, even though it was never seen during training.

To generate meaningful TFs, first, a density histogram is computed and then a Gaussian Mixture Model (GMM) is used to cluster densities in an unsupervised manner. GMMs have been previously used to cluster two-dimensional feature points [9], e.g., density and gradient magnitude. Our approach follows the same idea to cluster one-dimensional feature points, i.e., density values. The GMM represents each cluster as a 1D Gaussian function with a certain mean, i.e., the cluster center, and standard deviation, i.e., the cluster spread. To determine the number of components of the GMM, several GMMs with different numbers of components are build and the one with the lowest Bayesian information criterion (BIC) [8] value is selected. BIC penalizes the number of components and prevents overfitting using many components.

After computing the GMM, the number of peaks of the TF is sampled uniformly between 3 and 5. The represented density for each peak is sampled from the computed GMM. Next, a width in density space is sampled uniformly from $[0.005, 0.03]$ and the opacity at that peak from $[0.1, 1.0]$. As colormaps, predefined colormaps from SciVisColor[1] are randomly sampled. The generation process is visualized in Figure 16.

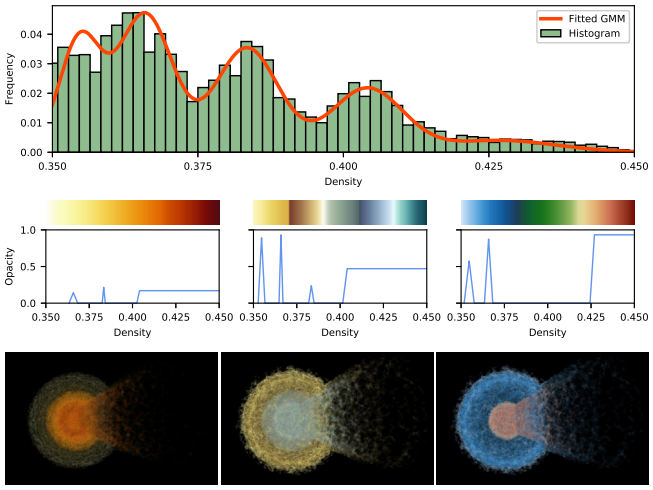


Fig. 16: First row: Histogram of the density values of the Ejecta data set and matched GMM. Second row: three sampled transfer functions with opacity and color. Third row: Renderings from the training data set with those transfer functions.

We note that it is important for the quality of the reconstruction that the color transfer functions in the training data include a broad range of colors. For example, if the training data only contains desaturated colors, strongly saturated colors during testing cannot be reconstructed.

Appendix D
Pull-Push Algorithm

As a baseline method to interpolate the sparse samples, we apply a variation of the push-pull algorithm [1, 3], see Alg. 1 for the pseudo code. The algorithm builds upon the idea of mip-map levels: first, the image is downscaled using bilinear interpolation with weights based on the mask. Then, the image is upscaled again and blended with the values at the finer levels with the mask values at the finer levels. We refer to Subsection 3.3 for more details in the context of the adaptive sampling pipeline. The pull-push algorithm can be directly extended to fractional masks as shown in Alg. 1. During the upsampling stage, the mask is not treated binary, i.e. either take the original pixel at the fine level or use the interpolated value from the coarse level, but fractional with a linear interpolation between the original value and the interpolated value. Furthermore, the algorithm consists only of linear pooling and interpolation layers which are easy to differentiate with respect to the input mask. We refer to Subsection E.3 for an outline on how to derive the backward pass.

Algorithm 1 Pseudocode of the pull-push algorithm for power-of-two input images (a version handling non-power-of-two inputs and the adjoint code for computing the derivative with respect to the mask and data are provided in the source code).

function INPAINTING(maskIn : HxW, dataIn : HxWxC)
  if H ≤ 1 or W ≤ 1 or all pixels are filled **then**
    return maskIn, dataIn ▷ end of recursion
  end if
  weighted area downsampling:
  maskLow, dataLow = zeros of shape $\frac{H}{2}$x$\frac{W}{2}$, $\frac{H}{2}$x$\frac{W}{2}$xC
  for $i, j \in \{0, ..., \frac{H}{2}-1\} \times \{0, ..., \frac{W}{2}-1\}$ **do**
    $N_{\max}, N_{\text{avg}}, d = \mathbf{0}^C$
    for $a, b \in \{2i, 2i+1\} \times \{2j, 2j+1\}$ **do** ▷ loop over neighbors in the fine grid
      $N_{\max} = \max\{N_{\max}, \text{maskIn}[a,b]\}$
      $N_{\text{avg}} \mathrel{+}= \text{maskIn}[a,b]$
      $d \mathrel{+}= \text{maskIn}[a,b] \cdot \text{dataIn}[a,b,:]$
    end for
    if $N_{\text{avg}} > 0$ **then**
      $\text{maskLow}[i,j] = N_{\max}$
      $\text{dataLow}[i,j,:] = d / N_{\text{avg}}$
    end if
  end for
  recursion:
  maskLow, dataLow = INPAINTING(maskLow, dataLow)
  weighted bilinear upsampling:
  maskOut, dataOut = zeros of shape HxW, HxWxC
  for $a, b \in \{0, ..., H-1\} \times \{0, ..., W-1\}$ **do**
    $N, W = 0, d = \mathbf{0}^C$
    $\hat{a} = a \div 2, \hat{b} = b \div 2$ ▷ Integer-division (round down), indices on the coarse grid
    $a', b' = -1$ if a, b is even else $+1$
    $N = \{(\hat{a}, \hat{b}, \frac{9}{16}), (\hat{a}+a', \hat{b}, \frac{3}{16}), (\hat{a}, \hat{b}+b', \frac{3}{16}), (\hat{a}+a', \hat{b}+b', \frac{1}{16})\}$
    for $(i, j, w) \in N \cap$ image **do** ▷ loop over neighbors if within bounds
      $N \mathrel{+}= w$ maskLow$[i,j]$
      $d \mathrel{+}= w$ maskLow$[i,j] \cdot$ dataLow$[i,j,:]$
      $W \mathrel{+}= w$
    end for
    $\text{maskOut}[a,b] = \text{maskIn}[i,j], \text{dataOut}[a,b,:] = \text{maskIn}[i,j] \cdot \text{dataIn}[i,j,:]$
    if $N > 0$ **then** ▷ blend interpolated values with original data
      $\text{maskOut}[a,b] \mathrel{+}= (1 - \text{maskIn}[i,j])\ N/W$
      $\text{dataOut}[a,b,:] \mathrel{+}= (1 - \text{maskIn}[i,j])\ d/N$
    end if
  end for
  return maskOut, dataOut
end function

1. https://sciviscolor.org/home/colormaps

APPENDIX E
DIFFERENTIATION OF THE SAMPLING AND RECONSTRUCTION STAGES

The adjoint code for the gradient propagation in the backward pass is automatically generated by PyTorch for the networks, the loss functions, and the sampling function Equation 5. For the pull-push algorithm (Alg. 1), the adjoint code was manually derived and implemented as a custom operation. In this section, we provide the fundamentals of the adjoint method to manually derive the adjoint code and show how it can be applied to the sampling function and the pull-push algorithm.

E.1 Fundamentals of the Adjoint Method

The adjoint method has a long history in Optimal Control Theory, we refer the interested reader to the book by Lions [4] for a complete mathematical introduction. Here, we briefly sketch the fundamentals following the notation by McNamara *et al.* [5].

By ignoring applications to linear systems and differential equations and focussing on chained functions instead, the adjoint method simplifies to an application of the chain rule. Let the algorithm be defined as a concatenation of functions f_i with parameters w_i starting from an input value x_0,

$$\begin{aligned} x_1 &= f_1(x_0, w_1) \\ x_2 &= f_2(x_1, w_2) \\ &\vdots \\ x_n &= f_n(x_{n-1}, w_n) \\ s &= J(x_n, w_J). \end{aligned} \tag{14}$$

The result s has to be a scalar value, this is crucial for the application of the adjoint method in this simple form. In the context of neural networks, x_0 would be the input image, f_1 to f_n the network layers with weights w_i and feature vectors x_i, J would be the loss function with target image w_J and s the scalar score.

During training, we are interested in the derivatives $\frac{\partial J}{\partial w_i}$ to update the weights or in e.g. $\frac{\partial J}{\partial x_0}$ to update the initial image in a feature-visualization context. First, given the – possibly vector valued – variables x_i and w_i, let the *adjoint variables* $\hat{x}_i$ and $\hat{w}_i$ be defined as the gradient of s with respect to x_i and w_i, $\hat{x}_i := \nabla_{x_i} s, \hat{w}_i := \nabla_{w_i} s$ as column vectors. Next, we drop the index i, as we require it to index the elements in the input and output vectors, and look at a single function $f \in \mathbb{R}^N \times \mathbb{R}^W \to \mathbb{R}^M$ with inputs $x \in \mathbb{R}^N, w \in \mathbb{R}^W$ and output $y \in \mathbb{R}^M$. The adjoint variables are then computed using

$$\hat{x} = J_{f,x}^T(x,w)\hat{y}\ , \quad \hat{w} = J_{f,w}^T(x,w)\hat{y}. \tag{15}$$

Here, the Jacobian matrix with respect to the different inputs is used, defined as

$$\left(J_{f,x}\right)_{ij} := \frac{\partial f_i}{\partial x_j}\ , \quad \left(J_{f,w}\right)_{ij} := \frac{\partial f_i}{\partial w_j}. \tag{16}$$

As one can see, given the adjoint variable of the output $\hat{y}$, the adjoint method propagates these gradients back through the derivatives of f to the adjoint variables of the inputs $\hat{x}$ and $\hat{w}$. In the context of the chained function Equation 14, this implies that, starting with gradients on the output $\hat{x}_n$ from the loss function, gradients are first propagated to $\hat{x}_{n-1}, \hat{w}_n$ via J_{f_i}, then to $\hat{x}_{n-2}, \hat{w}_{n-1}$, and so on until $\hat{x}_0, \hat{w}_1$ is reached.

To provide custom differentiable operations, two functions have to be provided: first, the forward code $y \leftarrow f(x,w)$ with input x and parameter w, and second, the backward code to compute $\hat{x}$ and $\hat{w}$ from $\hat{y}$, possibly using x, w from the forward pass again to compute the Jacobian.

E.2 Backward Pass of the Sampling Function

Using the theory above, we now present the adjoint code for the differentiable sampling from Subsection 3.2. This serves to highlight what is differentiated and how the gradients are propagated. Note that these functions are implemented based on PyTorch functions, PyTorch can automatically compute the derivatives.

The differentiable sampling stage takes the importance map I as input and produces the image of sparse samples S. In the framework of Equation 14, this can be seen as block of functions that is cut out in the middle. As parameters, the target mean μ and lower bound l, the sampling steepness α, the sample pattern P and the target image T are used. Note that no optimization with respect to these parameters is performed, their respective adjoint variables are unused. To recapitulate, the sampling is performed using the following steps:

$$\mu_I = \frac{1}{WH}\sum_{ij} I_{ij}\ , \quad I^{(1)} = I \tag{17a}$$

$$I'_{ij} = \min\left\{1, l + I^{(1)}_{ij}\frac{\mu - l}{\mu_I + \varepsilon}\right\} \tag{17b}$$

$$S_{ij} = \operatorname{sig}(\alpha(I'_{ij} - P_{ij}))T_{ij} \ \text{ with } \operatorname{sig}(x) = \frac{1}{1+e^{-x}}. \tag{17c}$$

Note that the second and third function act on each pixel ij of the images independently. Therefore, we use them as per-element functions to simplify the notation of the derivatives. Using the matrix notation from Equation 15, this would imply a diagonal Jacobian. Furthermore, T_{ij} and S_{ij} return the vector of channels at the specified location. In order to stay within the presented framework of the adjoint method, if variables are used by a function and later again by another function, these variables are passed through as additional outputs ($I^{(1)} = I$).

For the backward pass, we are given the gradients of the output $\hat{S}$ from the backward pass of the reconstruction. This equates to $\hat{x}_n$ in Equation 14. Then the gradients are propagated through the sampling algorithm in reverse order:

$$\begin{aligned} \hat{T}_{ij} &= \operatorname{sig}(\alpha(I'_{ij} - P_{ij}))\hat{S}_{ij} \\ \hat{P}_{ij} &= -\left(\alpha T_{ij} \operatorname{sig}'\left(\alpha(I'_{ij} - P_{ij})\right)\right)^T \hat{S}_{ij} \\ \hat{I'}_{ij} &= \left(\alpha T_{ij} \operatorname{sig}'\left(\alpha(I'_{ij} - P_{ij})\right)\right)^T \hat{S}_{ij} \\ &\text{with } \operatorname{sig}'(x) = \frac{d}{dx}\operatorname{sig}(x) = \operatorname{sig}(x)\operatorname{sig}(-x) \end{aligned} \tag{18a}$$

$$\hat{I}^{(1)}_{ij} = \begin{cases} \frac{\mu - l}{\mu_I + \varepsilon}\hat{I'}_{ij}\ , & I'_{ij} < 1 \\ 0\ , & I'_{ij} \geq 1 \end{cases}$$

$$\hat{\mu}_l = \sum_{ij}\left(\begin{cases} \frac{I^{(1)}_{ij}(l-\mu)}{(\mu_l+\varepsilon)^2}\hat{I'}_{ij}\ , & I'_{ij} < 1 \\ 0\ , & I'_{ij} \geq 1 \end{cases}\right) \tag{18b}$$

derivatives for μ, l, ε are omitted

$$\hat{I}_{ij} = \hat{I}^{(1)}_{ij} + \frac{1}{WH}\hat{\mu}_l \tag{18c}$$

E.3 Backward Pass of the Pull-Push Algorithm

As one can see in the previous section, deriving the adjoint code is done mechanically by deriving each line of code with respect to the inputs. This, however, produces a vastly longer code, therefore, we only outline the steps to derive the adjoint code of the pull-push algorithm Alg. 1. The full source code is available in the online repository.

The algorithm is a recursive algorithm with three stages: the downsampling to the coarse level, the recursive call, and the upsampling and interpolation with the fine level. During the backward pass, the order is reversed. First, the adjoint of the upsampling and interpolation at the finest level. Then the adjoint of the recursive call, which itself is the adjoint of upsampling, recursion, and downsampling. And lastly the adjoint of the downsampling.

REFERENCES

[1] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 43–54, 1996.

[2] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, Dec. 1964. doi: 10.1145/355588.365104

[3] M. Kraus. The pull-push algorithm revisited. *Proceedings GRAPP*, 2:3, 2009.

[4] J. L. Lions. *Optimal control of systems governed by partial differential equations*. Springer, 1971.

[5] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Transactions On Graphics (TOG)*, 23(3):449–456, 2004.

[6] M. Roberts. The unreasonable effectiveness of quasirandom sequences. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/, 2020. Accessed: 2020-02-14.

[7] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241. Springer International Publishing, Cham, 2015.

[8] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[9] Y. Wang, W. Chen, J. Zhang, T. Dong, G.-Y. Shan, and X. Chi. Efficient volume exploration using the gaussian mixture model. *Visualization and Computer Graphics, IEEE Transactions on*, 17:1560 – 1573, 12 2011. doi: 10.1109/TVCG.2011.97

[10] 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.

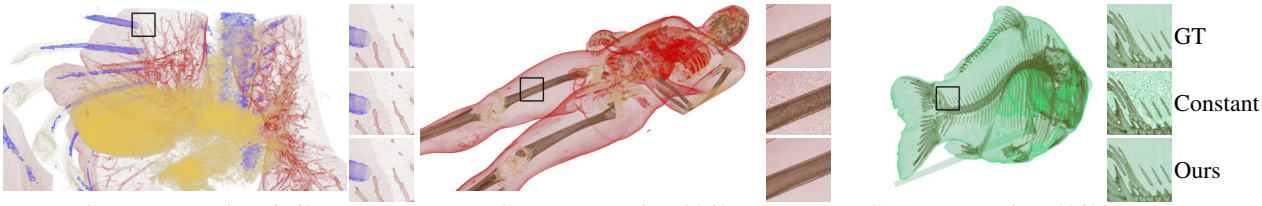
Appeared in EuroVis 2021 - Short Papers (Open Access).

This record was published online in this final form at https://diglib.eg.org/handle/10.2312/evs20211051. DOI: http://dx.doi.org/10.2312/evs.20211051

Analytic Ray Splitting for Controlled Precision DVR

Sebastian Weiss and Rüdiger Westermann

Technical University of Munich, Germany



Constant step size: 6760ms
Ours: 772ms

Constant step size: 9060ms
Ours: 534ms

Constant step size: 54600ms
Ours: 249ms

Figure 1: *DVR via post-classification. Using a constant step size (i.e.,* 0.2 *voxels) leads to sampling artifacts for high-frequency transfer functions. Analytic ray splitting (Ours) produces an image indistinguishable from the ground truth (GT) rendering, i.e., on an 8 bit color display the results are the same, and renders at less time than constant stepping.*

Abstract

For direct volume rendering of post-classified data, we propose an algorithm that analytically splits a ray through a cubical cell at the control points of a piecewise-polynomial transfer function. This splitting generates segments over which the variation of the optical properties is described by piecewise cubic functions. This allows using numerical quadrature rules with controlled precision to obtain an approximation with prescribed error bounds. The proposed splitting scheme can be used to find all piecewise linear or monotonic segments along a ray, and it can thus be used to improve the accuracy of direct volume rendering, scale-invariant volume rendering, and multi-isosurface rendering.

CCS Concepts

• ***Computing methodologies*** → ***Volumetric models;*** *Parallel algorithms;* • ***Mathematics of computing*** → *Quadrature;*

1. Introduction

In direct volume rendering (DVR), the perceived lightness is determined by considering an optical model, e.g., an emission-absorption model [DCH88, Lev88, Max95], and computing a numerical approximation of the resulting low-albedo volume rendering integral along the rays of sight [WM92]. In previous works, error bounds for the used numerical integration rules have been investigated [NA92, WM92, dBGHM97, CCdF15]. Etiene *et al.* [EJR+13] study the relationships between integration step size and accuracy of the results, and they propose a framework to assess the convergence of DVR algorithms with respect to step size, pixel size, and grid resolution.

Novins and Arvo [NA92] assume that both an emission and absorption field is given, i.e., the initial data values are pre-classified via transfer functions (TFs). When using trilinear interpolation in the cubical cells of a voxel grid, in each cell the profile of the interpolated quantities along a ray become cubic polynomials [PSL+98]. For these polynomials, error bounds for the numerical integration have been derived. For isosurface raycasting, Parker *et al.* [PSL+98], Neubauer *et al.* [NMHW02], and Marmit *et al.* [MKW+04] propose exact cell-wise ray splitting schemes based on the zero crossings of the trilinear interpolant. These methods provide algebraic solutions for ray-isosurface intersections in the trilinear interpolant.

When post-classification is used, i.e., the initial data values are first interpolated and then mapped to emission and absorption via a TF, the error bounds derived by Novins and Arvo break down. For post-classification and a linear variation of the optical properties within each cell, Williams and Max show that the integral can be computed algebraically [WM92]. Pre-integration [RKE00, EKE01] builds upon the assumption that the data values between adjacent sample points along a ray vary linearly. Then, integrals—including post-classification—can be pre-computed and used depending solely on the values at the sample points. Kniss

DOI: 10.2312/evs.20211051

Appeared in EuroVis 2021 - Short Papers (Open Access).

This record was published online in this final form at https://diglib.eg.org/handle/10.2312/evs20211051. DOI: http://dx.doi.org/10.2312/evs.20211051

et al. [KIL+03] extend on this finding by demonstrating that under the same assumption of piecewise linearity, multi-dimensional Gaussian TFs can be algebraically integrated. Scale-invariant volume rendering by Kraus [Kra05] addresses ray splitting in data space. The volume rendering integral is split into segments of equal length over which a piecewise monotonic change of the data values is assumed.

In the common case where piecewise linear 1D TFs are applied to the trilinearly interpolated data values, a ray is split into multiple piecewise cubic segments (see Figure 2). Then, piecewise linearity or monotony between adjacent sample points can only be justified for very small step sizes. In general, false classifications can occur and segments might even be missed entirely if the step size does not adapt to the segment boundaries.

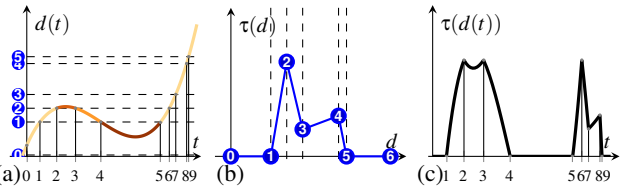


Figure 2: *(a) In a cell with trilinear interpolation, the data values along a ray $s(t)$ are described by a cubic polynomial. (b) A piecewise linear TF. (c) Applying the TF from (b) to the data values in (a) results in a piecewise cubic function. Dashed lines indicate the data values where the TF control points (blue) are hit.*

Contribution We propose an algorithm for adaptive step size control in DVR of post-classified data on a voxel grid. For post-classification, we consider piecewise linear TFs (Figure 2b). For cubical cells with trilinear interpolation, this splits a ray into multiple piecewise cubic segments. By determining these segments analytically, adaptive step size control is achieved. In particular, we

- build upon the algorithm by Marmitt *et al.* to split a ray through a voxel grid at all control points of a piecewise defined TF (subsection 2.1),
- apply controlled-precision numerical integration of post-classified data to solve the volume rendering integral on a per-segment basis (subsection 2.2),
- demonstrate the use of the proposed algorithm to split a ray into piecewise monotonic segments that are required by scale-invariant DVR (subsection 2.3).

All operations involved in analytic ray splitting have been implemented on the GPU to provide interactive frame rates even for large data sets. We have made our implementations available on GitHub†, so that all findings can be reproduced. A video showing the method and results in animation is available on YouTube‡.

2. Solving the Volume Rendering Integral

We assume a low-albedo emission-absorption model for volume rendering [Max95]. Let $\tau : [0,1] \to \mathbb{R}_0^+$ be the absorption due to a given density v along a ray $s(t) - d(t) = v(s(t))$ –, and $C : [0,1] \to \mathbb{R}_0^+$ the assigned color, both specified via a TF. With $g(v) = \tau(v)C(v)$ being the self-emission, the light intensity reaching the eye along the ray segment from $t = a$ to b is computed as

$$L(a,b) = \int_a^b g(v(s(t))) \exp\left(-\int_a^t \tau(v(s(u)))\mathrm{d}u\right)\mathrm{d}t. \quad (1)$$

We assume that the data values are given at the vertices of a voxel grid v. Within a cell, the data values are trilinearly interpolated, so that the data profile $v(s(t))$ along a ray through a cell is a cubic polynomial [PSL+98]. As a consequence, there are between zero and three locations along a ray in a single cell where a selected data value θ can be hit. These locations are given by the roots of $v(s(t)) - \theta = 0 \in [t_{\text{in}}, t_{\text{out}}]$. Marmitt *et al.* [MKW+04] and Neubauer *et al.* [NMHW02] have proposed numerical schemes to extract these roots. To avoid catastrophic cancellation when isolating the extrema of the cubic polynomial via the roots of its derivative, we use a numerically stable formulation [PTVF88, p. 184].

For cell-by-cell ray marching, we use the voxel traversal algorithm by Józsa *et al.* [JTC14], a stable reformulation of the algorithm by Amanatides *et al.* [AW+87, Hoe16]. Both the accumulation of numerical errors and errors due to rounding are reduced, so that even for large data sets the ray traversal routine does not introduce any perceivable errors. The algorithm provides the sequence of visited cells in front-to-back order, as well as the per-cell entry and exit points $[t_{\text{in}}, t_{\text{out}}]$. Note that this algorithm reports multiple roots either as no intersection or two separate intersections with two single roots, eliminating special handling of this case.

2.1. Transfer Function-Based Ray Splitting

In contrast to previous works, our proposed ray splitting scheme needs to consider the mapping of data values via a TF. We assume that the TF is given as a function satisfying two constraints: First, it is defined piecewise, i.e., specified by a finite number of control points with closed-form interpolation in between. This allows splitting the ray in data space at the control points so that no peaks are missed. Second, the absorption between two control points $\tau(v(s(u)))$ has to be analytically integrable. This is required to evaluate the inner integral in Equation 1 analytically and, thus, reduce the nested volume rendering integral to a single integral. The latter constraint holds, e.g., for all cell-wise polynomial interpolations. Specifically, we assume piecewise linear TFs as shown in Figure 2b. Absorption and emission values are given at a discrete set of N data values $0 = d_1 < d_2 < \ldots < d_N = 1$. We call these the TF control points. Each control point i stores the absorption τ_i and color C_i, with linear interpolation in between. If absorption and color are given as separate piecewise linear functions, they are combined into a single piecewise function in a pre-processing step.

Given the TF, all points where the ray takes on the values of the TF control points need to be found.We subsequently call these points, solutions of the cubic polynomial $d(t) - d_i$, the split points along a ray. For example, Figure 2 illustrates a situation where control point ❶ has three intersections at t_1, t_4, t_5, and control point ❸ has one intersection at t_7. Thus, multiple segments can occur within a single cell, and these segments need to be sorted efficiently. Note

† https://github.com/shamanDevel/Ray-Splitting-for-DVR
‡ https://youtu.be/bOLqIJd6dsw

Appeared in EuroVis 2021 - Short Papers (Open Access).

This record was published online in this final form at https://diglib.eg.org/handle/10.2312/evs20211051. DOI: http://dx.doi.org/10.2312/evs.20211051

that depending on the data values at the cell vertices and the number and width of the selected TF, up to $3N$ intersections can occur.

Furthermore, ambiguous cases regarding the occurrence of intersections need to be resolved. Such cases occur if no other intersection is in between two intersections for the same control point. Figure 3 illustrates all possible cases regarding the order in which these intersections can occur. Cases IIIa and IIIb are ambiguous since just from the data values at two consecutive intersections along a ray the order cannot be established. Such a case occurs in Figure 2a, where control point ❷ has intersections at t_2 and immediately again at t_3. To modulate the density polynomial with the transfer function, however, it is important to know if the segment from t_2 to t_3 visits the TF segment ❷–❸ (case IIIa) or ❶–❷ (case IIIb).

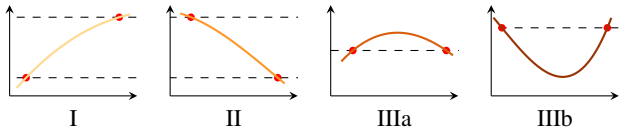


Figure 3: *Four cases of consecutive isosurface intersections.*

Sorting all computed split points along a ray can be realized in $O(N' \log N')$ operations using standard sorting algorithms, where N' is the number of control points falling into the data range covered by the current cell. Ambiguous cases can be decided, in principle, by evaluating the polynomial in the middle of the interval between two consecutive split points, and testing whether the value is greater or smaller than the data value at the first point. This approach, however, requires an additional evaluation of the polynomial and should be avoided. The following adaptive ray splitting algorithm can iterate over the split points in $O(N')$ operations, and handles the ambiguous cases directly without extra evaluations.

First, the data value at the cell entry point (t_0) is evaluated. Then, the TF segment (d_i, d_{i+1}) that contains this density value is found via binary search. In the example in Figure 2, this is the segment ❶–❶. Now due to continuity, the next split point can only be at the data values corresponding to these two control points. W.l.o.g. let the next split point at t_1 be at the data value of control point ❶, the case for the lower control point d_i follows similarly by symmetry considerations. Since this data value is approached from a lower data value, the next intersection at t_2 can only be at the same data value at control point ❶ again or at the next larger data values at control point ❷. The latter situation is shown in the example. Similarly, for t_3 the data values at control points ❷ and ❸ are tested, but now the data value at ❷ is visited again. This indicates case IIIa and the search direction is changed. The next split can now only occur according to the data value at ❶ (case II) or ❷ (case IIIb), as we approach them from a larger data value. This process is repeated until all split points are processed and the ray exits the cell at t_{exit}. For each ray segment that is computed, the volume rendering integral is solved via quadrature, and the resulting color values are blended in front-to-back order.

2.2. Controlled Precision DVR

Novins and Arvo [NA92] propose various quadrature schemes for a single cell and a single linear transfer function. For a piecewise

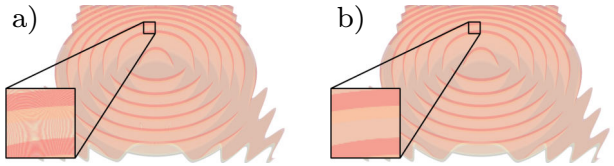


Figure 4: *a) Scale-invariant DVR with a step size of* 0.1 *voxels suffers from artifacts due to under-sampling and erroneously assumed per-segment monotony. b) Analytic ray splitting finds the exact locations of piecewise monotonic segments along the rays.*

TF, however, their method, cannot be directly applied to the entire cell. Instead, using our proposed ray splitting algorithm, smaller segments within a cell bounded by the control points of the TF are extracted. For each such segment, we apply the algorithm by Novins and Arvo to evaluate the integral per segment and accumulate the results over the segments and cells. In the benchmarks, we use Simpson quadrature with a fixed number of 10 quadrature points to reduce branch divergence in CUDA. Using more quadrature points does not improve the results.

2.3. Scale-Invariant DVR

Kraus [Kra05] introduced a model for scale-invariant volume rendering, in which the volume rendering integral in physical space is replaced by the scale-invariant integral in data space. This can be seen as the limit case where infinitely many semi-transparent isosurfaces are blended (Figure 4). This model builds upon the assumption that the density field is piecewise monotonic within the interval $[a, b]$ in data space, an assumption that is only fulfilled at small step sizes.

In our framework, scale-invariant DVR including an adaptive step size that splits the data into piecewise monotonic segments can be easily realized. To enforce monotony over each integration interval, additional split points need to be included at the extrema of the density profiles along the ray segments. In the root finding method by Marmitt *et al.*, these split points are computed in-turn to split a ray segment into sub-segments in which only zero or one root of the polynomial is located.

3. Evaluation on Synthetic Datasets

We compare the quality and performance of adaptive ray-splitting to DVR via ray-casting with constant step size and pre-integrated DVR [RKE00, EKE01] with a 32 bit floating point table of size 512^2. The Marschner Lobb dataset [ML94] on a 128^3 voxel grid is rendered to a 512^2 viewport. Two triangular TFs with three peaks of width 0.05 and 0.002, respectively, are used to analyze the sensitivity of the DVR variants to the characteristics of the mapping function. To generate ground truth images, we use constant stepping with a step size of 0.0001 voxels using double-precision floats.

To obtain insights as to where approximation errors occur, the Marschner Lobb is rendered using different DVR algorithms (see Figure 5). At a peak width of 0.05, the transitions between the features in the image are smooth, yet constant stepping already shows

Appeared in EuroVis 2021 - Short Papers (Open Access).

This record was published online in this final form at https://diglib.eg.org/handle/10.2312/evs20211051. DOI: http://dx.doi.org/10.2312/evs.20211051

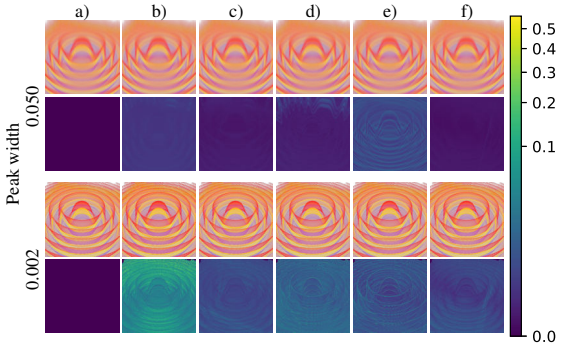


Figure 5: *Comparison of different DVR algorithms. (a) Ground truth, (b,c) and (d,e) constant stepping and pre-integration, respectively, with a step size of 0.1 (b,d) and 0.01 (c,e) voxel, (f) adaptive step size with Simpson quadrature. The first and third row show the rendering, the second and forth row the per-pixel L_2-norm of the color difference to the baseline, scaled using a power norm of $x^{0.3}$.*

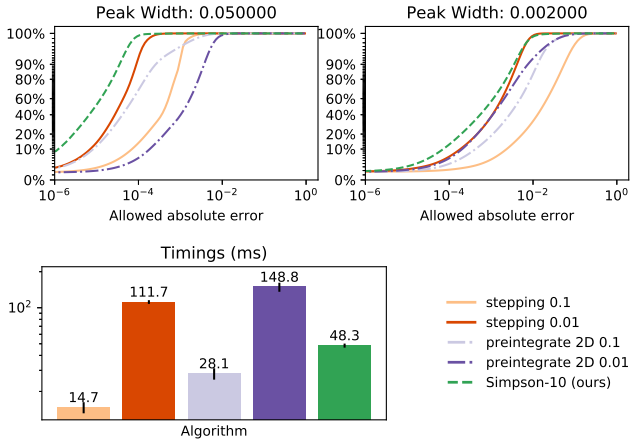


Figure 6: *Accuracy statistics and timings for the Marschner Lobb.*

noticeable pixel errors over the whole image. Analytic ray splitting with Simpson quadrature produces significantly lower numerical errors below the perceivable tolerance. When narrowing the peaks to 0.002, the errors introduced by constant stepping become unacceptably high. Constant stepping with a step size of 0.1 fails to resolve the thin structures, which can be improved to some degree by decreasing the step size to 0.01. Pre-integration quality is affected by sampling artifacts, discretization errors in the pre-integration table, and interpolation accuracy in the GPU texture units. Adaptive ray splitting generates results that are visually indistinguishable from the ground truth rendering, and it seems en par with constant stepping with a step size of 0.01 voxel. However, Figure 6 shows so-called regression error characteristic (REC) curves, which indicate the percentage of pixels within a certain allowed error. It can be seen that ray splitting achieves higher accuracy than all alternatives. Table 1 shows the error metrics for the images in Figure 1.

All DVR algorithms have been implemented in CUDA, and performance measures were taken on a NVidia RTX 2070 GPU. Timings are averaged over 10 different frames of resolution 512^2, by moving the camera randomly around the datasets. No acceleration structure is used. The timings in Figure 6 show the linear performance scale of constant stepping approach in the step size. Adaptive ray splitting lies between constant stepping with a step size of 0.1 and 0.01. Profiling shows that by far the most time is spent in the solve step—to accurately determine the points along the ray where the data values selected by the TF control points are taken—and the procedure that builds the piecewise polynomials.

Dataset	Thorax	Human	Carp
Simpson *e*	4.69e-5	1.13e-4	6.60e-4
Simpson *t*	7.72e2	5.34e2	2.49e2
Stepping *s*	2.14e-3	1.56e-2	4.93e-4
Stepping *e*	3.24e-4	7.47e-4	6.08e-4
Stepping *t*	6.76e3	9.06e3	5.46e4

Table 1: *Error and timing statistics for the data sets in Figure 1. s, e and t, respectively, indicate step size in constant stepping, mean square pixel error compared to the baseline rendering, and rendering time in milliseconds.*

4. Real-World Datasets

We further evaluate the quality and performance of the DVR algorithms on three CT scans (Figure 1): A human thorax at a size of $512^2 \times 286$, a full scan of the human body at a size of $512^2 \times 1884$, and a carp at size 512^3, rendered at resolution 1920×1080. We compare the ray-splitting algorithm with Simpson quadrature to ray tracing with a constant step size and measure the mean absolute error to the baseline and the execution time. For constant stepping, the step size is halved until the rendering has converged, i.e. the output does not change within an error of $1/256$ per pixel anymore. Table 1 reports the final step size, timings and error to the baseline. As one can see, adaptive ray splitting is 1-2 magnitudes faster at a lower or similar error than converged constant stepping.

5. Conclusion

We have presented an algorithm that analytically splits rays through a post-classified emission-absorption volume at the data values given by the control points of a piecewise linear transfer function. This allows for solving analytically the absorption integral and numerically the emission integral up to a user-defined precision. Our evaluations have shown that the performance of analytic ray splitting can be even higher than that of constant stepping when triangular TFs with rather narrow peaks are used. We consider the proposed renderer as baseline for comparative purposes as well as a framework to integrate alternative rendering options such as scale-invariant DVR and multi-isosurface rendering.

In the future, we will in particular investigate the use of analytic ray splitting for differentiable volume rendering including post-classification. Ultimately, we plan to develop algorithms for converting physical fields in situ into a compact latent code that can be interpreted by a renderer to produce a meaningful visual representation. This requires differentiable renderers that can compute per-pixel derivatives with respect to the post-classification process.

References

[AW+87] J. Amanatides, A. Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[CCdF15] 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*, pages 17–24. IEEE, 2015.

[dBGHM97] M. W. de Boer, A. Gröpl, J. Hesser, and R. Männer. Reducing artifacts in volume rendering by higher order integration. *IEEE Visualization'97 Late Breaking Hot Topics*, pages 1–4, 1997.

[DCH88] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4):65–74, 1988.

[EJR+13] T. Etiene, D. Jönsson, T. Ropinski, C. Scheidegger, J. L. Comba, L. G. Nonato, R. M. Kirby, A. Ynnerman, and C. T. Silva. Verifying volume rendering using discretization error analysis. *IEEE transactions on visualization and computer graphics*, 20(1):140–154, 2013.

[EKE01] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '01, page 9–16, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/383507.383515.

[Hoe16] R. K. Hoetzlein. GVDB: Raytracing Sparse Voxel Database Structures on the GPU. In U. Assarsson and W. Hunt, editors, *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2016. doi:10.2312/hpg.20161197.

[JTC14] P. Józsa, M. J. Tóth, and B. Csébfalvi. Analytic isosurface rendering and maximum intensity projection on the gpu. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Václav Skala-UNION Agency, 2014.

[KIL+03] J. Kniss, M. Ikits, A. Lefohn, C. Hansen, E. Praun, et al. Gaussian transfer functions for multi-field volume visualization. In *IEEE Visualization, 2003. VIS 2003.*, pages 497–504. IEEE, 2003.

[Kra05] M. Kraus. Scale-invariant volume rendering. In *VIS 05. IEEE Visualization, 2005.*, pages 295–302. IEEE, 2005.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer graphics and Applications*, 8(3):29–37, 1988.

[Max95] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[MKW+04] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing. In *VMV*, volume 4, pages 429–435, 2004.

[ML94] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings Visualization'94*, pages 100–107. IEEE, 1994.

[NA92] K. Novins and J. Arvo. Controlled precision volume integration. In *Proceedings of the 1992 workshop on Volume visualization*, pages 83–89, 1992.

[NMHW02] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. Cell-based first-hit ray casting. In *VisSym*, pages 77–86, 2002.

[PSL+98] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 233–238. IEEE, 1998.

[PTVF88] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in c, 1988.

[RKE00] S. Rottger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*, pages 109–116, 2000. doi:10.1109/VISUAL.2000.885683.

[WM92] P. L. Williams and N. Max. A volume density optical model. In *Proceedings of the 1992 workshop on Volume visualization*, pages 61–68, 1992.

Differentiable Direct Volume Rendering

Sebastian Weiss* and Rüdiger Westermann†

Technical University of Munich

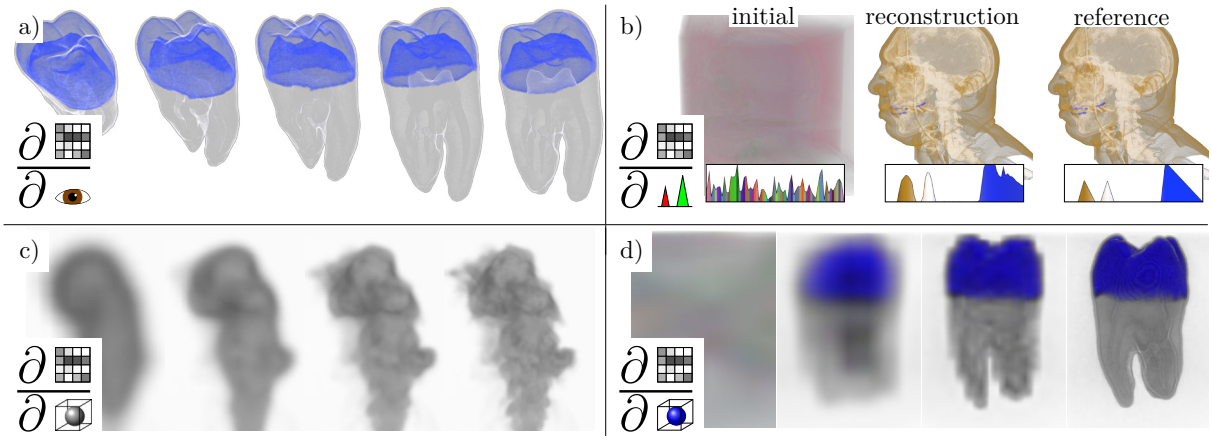


Fig. 1. A fully differentiable direct volume renderer is used for a) viewpoint optimization, b) transfer function optimization, and optimization of voxel properties using c) an absorption-only model and d) an emission-absorption model with rgbα transfer functions. a), c) and d) show intermediate results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

*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

structures. A follow-up work [36] addresses this issue but limits the differentiability to volume densities and colors.

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: uv the pixel positions where to shoot the rays

$color_i=0$ ▷ initial foreground color
$x_o,\boldsymbol{\omega}=f_{\text{camera}}(\mathit{uv},cam)$ ▷ 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

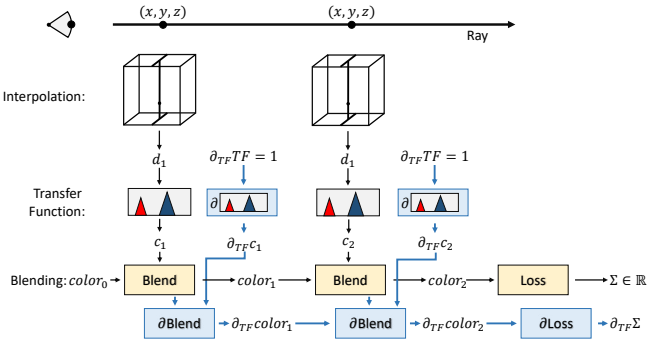


Fig. 2. Schematic representation of the forward method for TF reconstruction. Gradients are stored in the forward variables (blue), and parameter values are propagated simultaneously.

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

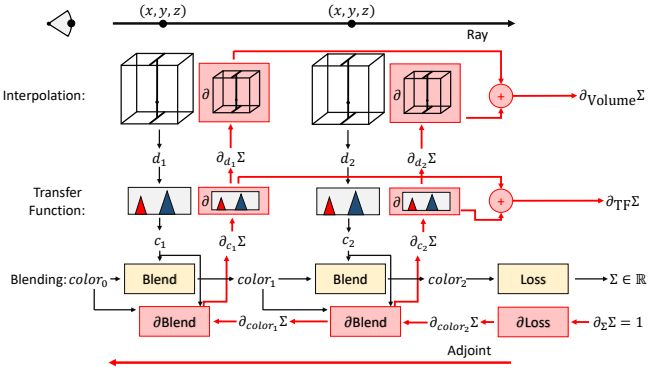


Fig. 3. Schematic representation of the adjoint method for density and TF reconstruction. Gradients in the adjoint variables (red) are propagated backward through the algorithm. A circled + indicates the summation of the gradients over all steps and rays.

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: $\hat{cam} \mathrel{+}= \partial f_{\text{camera}}(uv, cam)^T [x_o; \omega]$
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.

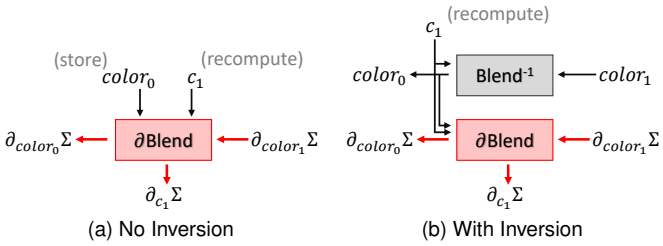


Fig. 4. (a) To compute the current contribution c_i, intermediate accumulated colors $color_i$ need to be stored for every step along the ray. (b) The inversion trick enables to reconstruct $color_i$ from $color_{i+1}$. Thus, only the final color used in the loss function needs to be stored.

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].

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2021.3114769

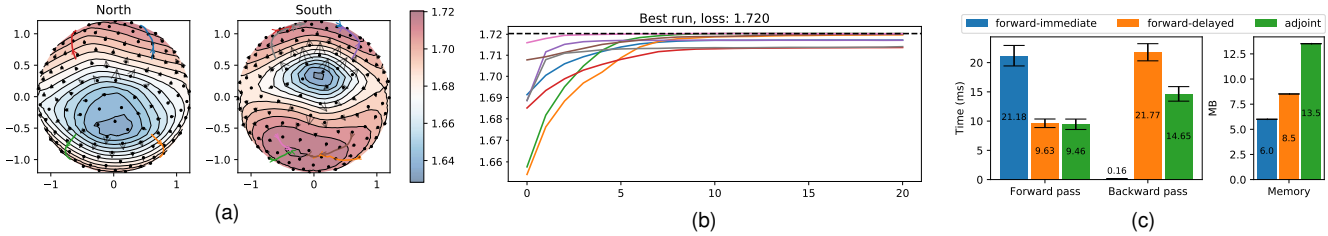


Fig. 5. Best viewpoint selection using maximization of visual entropy. The tooth dataset (Fig. 1a) is rendered from different viewpoints on a surrounding sphere. (a) Color coding of loss values for viewpoints on the northern and southern hemispheres, with isocontours (black lines) of the loss and local 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.

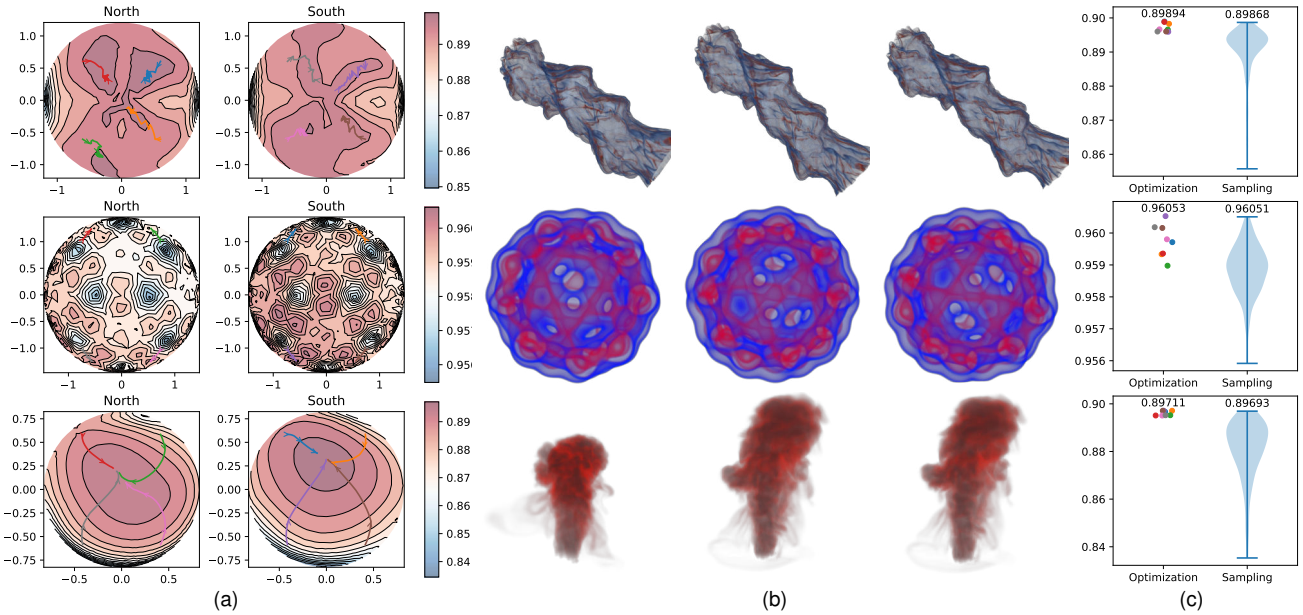


Fig. 6. Best viewpoint selection using maximization of visual entropy for datasets jetstream (256^3), potential field of a C60 molecule (128^3), and smoke plume (178^3). Comparison of DiffDVR with eight initializations against random uniform sampling over the sphere of 256 views. (a) Optimization paths 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

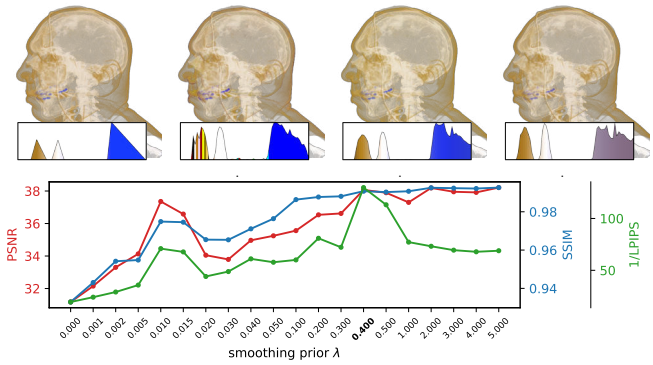


Fig. 7. Effect of the smoothing prior (Equation 12). A small value of λ leads to "jagged" TFs which can accurately predict small details like the teeth in blue but introduce low frequency color shifts resulting in low PSNR and SSIM [52]. A large smoothing prior smooths out small details.

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

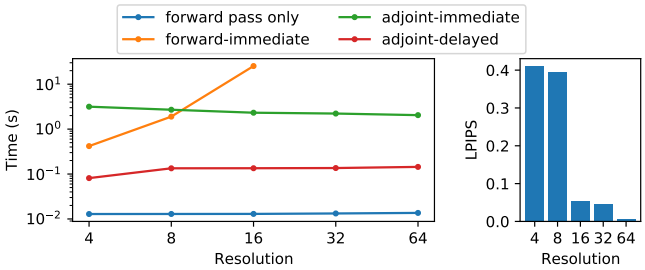


Fig. 8. Timings and loss function values for different AD modes and resolutions of the reconstructed TF. Timings are with respect to a single epoch.

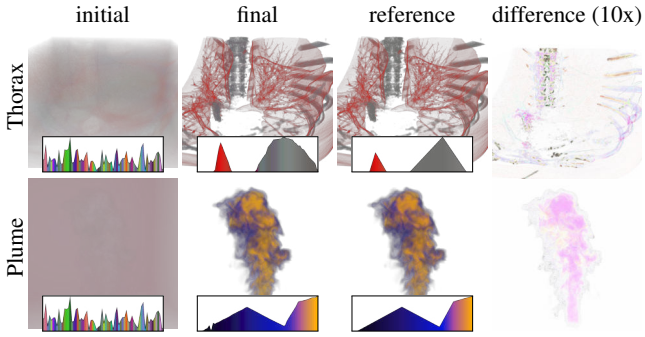


Fig. 9. TF reconstruction using a CT scan of a human thorax (PSNR=42.6dB) and a smoke plume (PSNR=47.8dB, SSIM=0.999). The used hyperparameters are the same as for the skull dataset. From left to right: Start configurations for the optimizer, optimized results, ground truths, pixel differences (scaled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

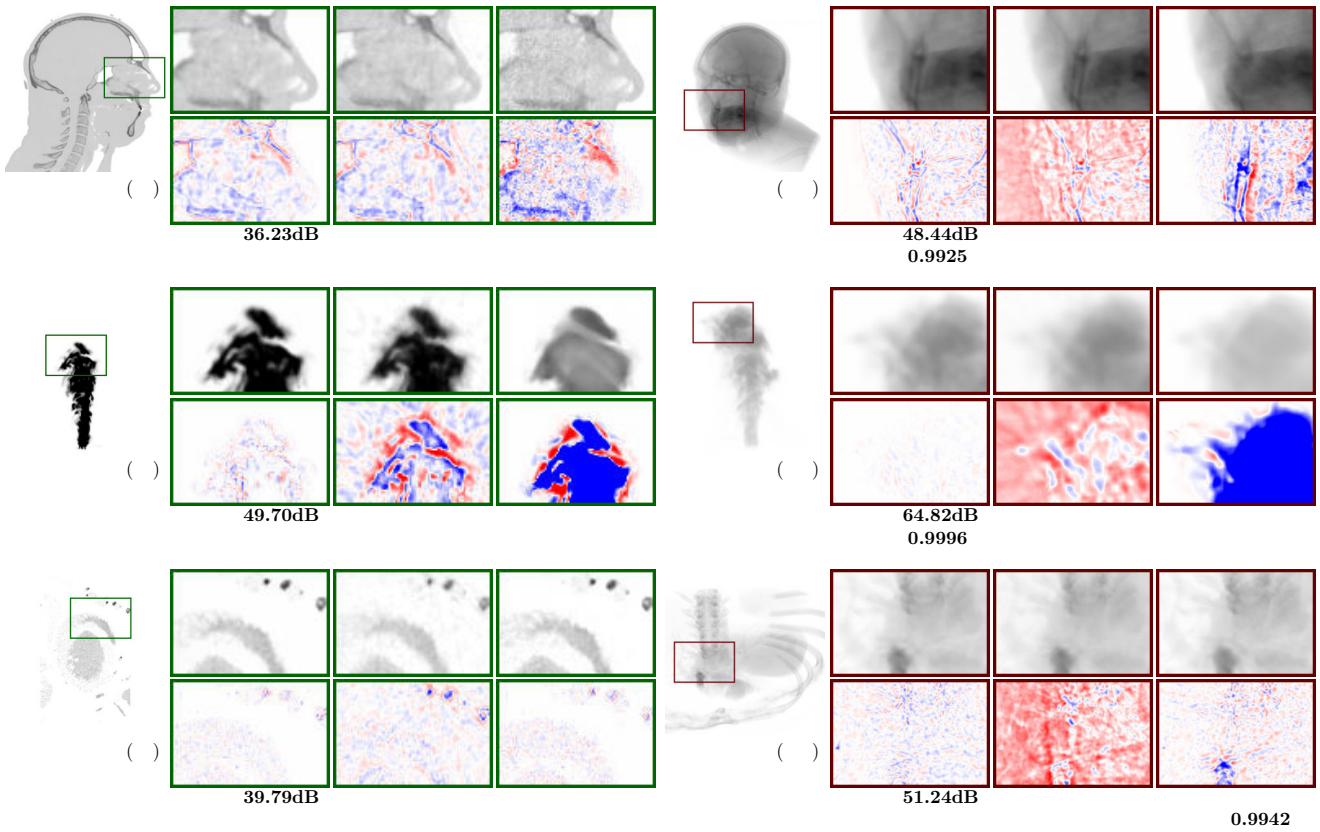


Fig. 10. Density reconstruction using an optical absorption-only model. Comparison between DiffDVR, algebraic reconstruction provided by the ASTRA-toolbox [49, 50] and Mitsuba's differentiable path tracer [36]. For each algorithm, a single slice through the center of the reconstructed volume and a volume 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.

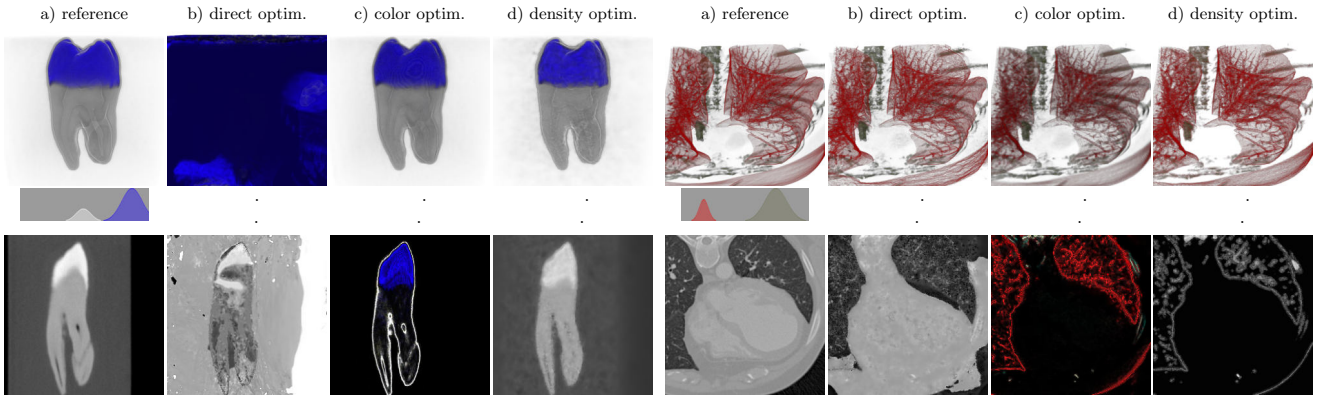


Fig. 11. Density optimization for a volume colored via a non-monotonic rgbα-TF using an emission-absorption model. (a) Rendering of the reference volume of a human tooth and a human thorax. (b) Local minimum of the loss function. (c) Pre-shaded color volume as initialization. (d) Final result of the 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).

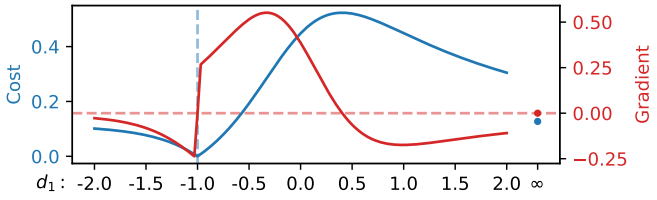


Fig. 12. 1D example for a density optimization with a Gaussian TF with the optimum at a density of -1.0. For a value > 0.4, the gradient faces away from the optimum.

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.