## Title

FoVolNet: Fast Volume Rendering using Foveated Deep Neural Networks

Journal
IEEE Transactions on Visualization and Computer Graphics, 29(1)

ISSN
1077-2626

Authors
Bauer, David
Wu, Qi
Ma, Kwan-Liu

Publication Date
2023

DOI
10.1109/tvcg.2022.3209498

Peer reviewed

Published in final edited form as:

*IEEE Trans Vis Comput Graph*. 2023 January ; 29(1): 515–525. doi:10.1109/TVCG.2022.3209498.

# FoVolNet: Fast Volume Rendering using Foveated Deep Neural Networks

**David Bauer**,

**Qi Wu**,

**Kwan-Liu Ma**

University of California at Davis.

## Abstract

Volume data is found in many important scientific and engineering applications. Rendering this data for visualization at high quality and interactive rates for demanding applications such as virtual reality is still not easily achievable even using professional-grade hardware. We introduce FoVolNet—a method to significantly increase the performance of volume data visualization. We develop a cost-effective foveated rendering pipeline that sparsely samples a volume around a focal point and reconstructs the full-frame using a deep neural network. Foveated rendering is a technique that prioritizes rendering computations around the user's focal point. This approach leverages properties of the human visual system, thereby saving computational resources when rendering data in the periphery of the user's field of vision. Our reconstruction network combines direct and kernel prediction methods to produce fast, stable, and perceptually convincing output. With a slim design and the use of quantization, our method outperforms state-of-the-art neural reconstruction techniques in both end-to-end frame times and visual quality. We conduct extensive evaluations of the system's rendering performance, inference speed, and perceptual properties, and we provide comparisons to competing neural image reconstruction techniques. Our test results show that FoVolNet consistently achieves significant time saving over conventional rendering while preserving perceptual quality.

**Index Terms—**

Volume data; volume visualization; deep learning; foveated rendering; neural reconstruction

## 1 Introduction

Since its beginnings, volume rendering has been an integral part of the scientific and biomedical visualization community. Over time, tremendous improvements have been made to the quality and performance of volume rendering algorithms. Yet, with advances in high-fidelity rendering comes increased computational cost. Many state-of-the-art techniques like path tracing or global illumination have outpaced the capabilities of consumer hardware,

davbauer@ucdavis.edu.

This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2022.3209498, provided by the authors.

putting these techniques out of reach for interactive applications. There are also other relevant issues in this area, such as data management and storage. However, the visualization community has already produced various methods to mitigate these problems.

For instance, prior work [3, 13, 33, 49] offers solutions for rendering extremely large volumes that do not fit the main memory of a system. Visualization of these data became viable through the introduction of streaming techniques such as out-of-core rendering which eliminate the need for the whole dataset to be present in memory. Such methods are a means of emancipation. They help us depend less on specific characteristics of the data—in this case, its size. When it comes to visual quality, state-of-the-art rendering techniques lack similar means. The higher the visual quality a technique produces, the more computational resources are generally needed to compute it. Although ongoing research has produced more efficient methods over the years, we are still bound by factors like the number of rays, sampling rates, or the type of illumination. Therefore, visualizing volume data using high-quality shading techniques at interactive framerates remains challenging—especially for demanding applications like immersive visualization.

This work pries open the tight coupling between rendering technique and computational cost. We introduce FoVolNet—a complete volume rendering pipeline that aims to loosen the relation between technique and cost. By skipping the majority of screen-space pixel processing and replacing it with constant-time neural image reconstruction, we can achieve drastic performance improvements without sacrificing visual quality. We take inspiration from literature on foveated rendering and deep learning based image denoising. Prior work [12, 52, 58] has shown that taking the human visual system (HVS) into account when rendering data can yield excellent results for performance without perceptible quality loss. Utilizing the characteristics of the HVS is a crucial part of our design, as it allows us to concentrate computational resources. Accordingly, FoVolNet renders sparse images with dense foveated areas. A neural image reconstruction network restores the missing visual information, allowing us to skip the majority of screen-space pixel processing and replacing it with a constant-time inference step. This makes it possible to visualize volumes in high quality at a much lower computational cost than conventional rendering methods. In turn, this decoupling allows us to achieve faster and more consistent frame rates in various rendering setups.

We conduct thorough tests involving the system's overall rendering performance, image quality, and other properties such as effective compression rate to evaluate our approach. The results show that FoVolNet faithfully reconstructs full frames at a fraction of the time it takes conventional methods to produce the same output.

## 2 Related Work

Our work is related to topics in volume rendering, optical flow estimation, and deep learning methods in computer graphics and image processing. In this section, we discuss related works in these fields.

### 2.1 Advanced Volume Rendering

Optical models for computing advanced illumination effects (e.g., ambient occlusion, global shadows, multi-scattering) in volume rendering were first outlined by Max in the late 1990s [36, 37]. However, since these models are generally expensive to compute, a large body of work has focused on how performance can be improved. The ambient occlusion model [6, 16, 28, 45–47, 51] simulates the occlusion effect within a small neighborhood of the sample point, estimating the local extinction within a small spherical region. More recently, deep neural networks have been used to generate ambient occlusion effects for volume rendering [7]. However, this model only accounts for local shadows and lacks cues for large-scale occlusions. Computing global shadows require considering attenuation between light sources and the sample points. To efficiently compute global shadows, many different approaches have been proposed, including half-angle slicing [24, 25], plane sweep [53], shadow volume [44], light volume [66], or voxel cone tracing [48]. However, these methods still cannot calculate realistic multi-scattering effects. More recently, the use of ray tracing presents a new trend of volume visualization algorithms that implement a highly realistic multi-scattering model [5, 27, 31, 40, 40]. By combining them with production ray-tracing software [57] and realistic BRDF classification techniques [19], unbiased volume rendering can finally be achieved. However, these algorithms are computationally costly when high-resolution data—which requires a high sampling rate—and complex lighting conditions are combined. Thus, the rendering performance of these methods can quickly deteriorate. In this work, we lift a sizeable portion of the computational burden that such techniques incur on modern hardware. We reduce the screen space sample count and replace the skipped computations with a constant-time neural network inference step.

### 2.2 Foveated Rendering

Recent advances in eye-tracking technology and the market push towards augmented and virtual reality applications have intensified research in foveated rendering techniques. Computational power is crucial for high-fidelity rendering applications and most of today’s immersive content. It is therefore of paramount importance to distribute resources efficiently. The capacity of the human visual system to perceive high levels of detail is limited to a relatively small focal area [1]. The fovea, which is the area of the visual field with the highest acuity, only makes up about 5.2 degrees around the optical axis of the eye [59]. Foveated rendering approaches use that fact by focusing computational resources on these areas. Guenter et al. [12] were one of the first to develop such an approach by rendering scenes in multiple levels of detail in concentric circles around the focus point. Later approaches [52] use variable sampling rates that prioritize the focal area. Weier et al. [58] combine this approach with frame reprojection to reduce peripheral flickering.

### 2.3 Deep Learning for Image Denoising

One of the prominent uses of deep learning in the computer graphics field is image denoising. It is the process of refining noisy images, which are usually the result of Monte Carlo (MC) renderings with a low number of samples per pixel (SPP). A primitive approach to improving image quality is to increase SPP. However, this method requires significantly longer processing times per frame. Recent work has leveraged deep learning to refine low SPP images without this computational overhead. Early approaches [2, 4] already achieve impressive results using CNNs. These works have established the two fundamental philosophies of image denoising in today's literature. On the one hand, there is direct prediction [4]. A method to produce denoised images as a direct result of network inference. On the other hand, kernel prediction [2] approaches use CNNs to produce image filters. The denoising operation is performed by applying these filters to the input image in a separate step.

Subsequent work followed in these footsteps, furthering the potential of these two concepts. Wong et al. [62] introduce residual connections for direct prediction networks to improve single-frame image quality. To the same end, Xu et al. [64] and Lu et al. [32] conducted experiments on using adversarial networks [11] to train direct prediction models. More recently, Hofmann et al. [17] have applied direct prediction to the domain of volume path tracing. Similarly, Weiss et al. [60] explore the utility of direct prediction for reconstructing adaptive volume ray marching. Along with works like Kettunen et al.'s gradient-space denoising [22] and Wong et al.'s ResNet approach [62], they investigate the effect of various auxiliary input features on final image quality. Following the kernel prediction path [2], we see work by Vogels et al. [56] who extend the approach by incorporating neighboring frames into training to facilitate temporal stability. Hasselgren et al. [15] build on this notion, creating temporally stable image sequences using predictive adaptive sampling and temporal blending. Gharbi et al. [10] solve problems involving motion blur and depth-of-field using a kernel prediction approach with splatting.

Neural architectures used in these projects vary. However, there are certain identifiable trends. The U-Net architecture [43], initially developed for medical image segmentation tasks, has proved to be a practical choice for denoising tasks. Many works [4, 10, 15, 17, 22] base their design on the U-Net's image encoder-decoder principle. Extensions often include skip connections and recurrent feedback, which tend to increase image quality and temporal stability. Other works [2, 32, 56, 62, 64] use more conventional CNN or RNN models. Interestingly, there is no apparent connection between chosen architecture (U-Net, CNN, RNN) and the denoising approach (direct prediction, kernel prediction). Several works [17, 32, 64] also use an additional critique network for their adversarial training. Named after its shape, the W-Net poses an extension to the U-Net and was introduced by Thomas et al. [55]. It comprises two U-Nets in sequence. One is used for feature extraction, while the other serves to generate and apply convolutional filters to the input. This design facilitates optimization through selective quantization without significant image quality loss. For further reading on this topic, we refer the interested reader to Huo et al.'s survey on deep-learning-based image denoising techniques [18]. Our network design is based on the W-Net architecture. We introduce a hybrid approach combining direct and kernel prediction to achieve the best results for sparse image inputs. This differs from conventional image denoising approaches in that missing visual information needs to be generated by the network. Therefore, pure kernel prediction networks are not suitable for this task as kernels only operate on existing pixel values. DeepFovea [21] is the current state-of-the-art for such sparse frame reconstructions using solely direct prediction. Our approach translates this initial idea to the domain of scientific rendering and significantly improves visual quality and performance.

### 2.4 Optical Flow

Perceived motion in video sequences results from incremental changes in the positions of objects in a scene or by camera movement. Estimating the optical flow of elements in adjacent frames is an active area of research, and various approaches have been proposed in recent years.

An early approach by Farnebäck et al. [9] introduces a motion estimation algorithm that characterizes pixel neighborhoods as polynomials and uses those to find a mapping between frames. They propose a multiscale approach that uses a priori motion estimation. This allows the algorithm to iterate and refine the estimation by considering differently sized search windows. This increases the robustness and quality of the results. Subsequent works [29, 41, 54] tend to emulate this iterative, hierarchical approach. Most recently, Hanika et al. [14] have introduced a method based on this scheme. Unlike previous approaches, this algorithm sacrifices some quality in favor of speed. It also manages disocclusions gracefully.

For our work, we utilize Hanika et al.'s approach [14] to reproject frames during training. By warping a previous frame, we can gain more visual information about the current image, which can be used to construct a loss function that requires the network to match reprojected frames [21]. This additional information helps increase image quality and supports retaining temporal stability between frames.

## 3 Methods

FoVolNet is a complete raymarching volume rendering pipeline that is supported by a neural network (Figure 2). The overall rendering process consists of two critical steps. First, the volume needs to be rendered. Instead of rendering the whole frame, we selectively render a subset of pixels. A neural network is then used to reconstruct the full frame from this subset. The following sections contain details on our approach. There, we discuss the implementation and design of each stage of FoVolNet.

### 3.1 Foveated Rendering

We implement a ray marching system that facilitates sparse, foveated rendering. The renderer is implemented in CUDA and OptiX and supports global ray marched shadows. For the shadow computation we cast one shadow ray per sample step towards the light source using $\frac{1}{4}$ of the main sample rate. This renderer allows us to reduce rendering time as overall pixel density decreases. Our foveated rendering technique is based on binary sample maps generated from noise patterns that determine which pixels in screen space should be sampled by the volume renderer.

**3.1.1 Noise Patterns—**The noise patterns used in this work (Figures 3, 4 (left)) can be tiled seamlessly which allows us to cover an arbitrarily large frame. In our experiments, we tested noise map tile sizes that ranged from 16 × 16 to 256 × 256 pixels; however, there was no noticeable difference in the final image quality.

A comparison of different sources of noise is shown in Figure 3. Using noise sampled from a uniform distribution, like the one shown on the left, can result in energy spikes across the pattern. This is generally not desirable as it causes samples in the sampling map to be unevenly distributed. The temporal mean of uniform noise exhibits similar problems. Blue noise (Figure 3 (b)) is rich in high frequencies and generally does not suffer from low-frequency energy spikes in the spatial domain. Its distribution closely models that of the visual receptors on our retina and is therefore ideal for creating perceptually unobtrusive sampling patterns. However, conventional blue noise suffers from temporal instability, as can be seen in Figure 3 (e).

We use spatio-temporal blue noise (STBN) [61] to generate temporally stable sample patterns while preserving the perceptual advantages of conventional blue noise (Figure 3 (c), (f)). As opposed to sequences of independent 2D blue noise or 3D blue noise patterns, STBN is not only blue in the spatial domain - every pixel is blue over time. This property is desirable since it helps our reconstruction network to produce stable and perceptually clean image sequences.

### 3.1.2 Sample Maps

To generate a sample mask $M$, we compare the noise value $N(u, v)$ at position $u$, $v$ against a threshold $\tau$. If $N(u, v) < \tau$ we set $M(u, v)$ to 1; otherwise, it is set to 0. The value for $\tau$ can be adjusted to vary the sampling density. We define the density of the base noise pattern as $P_b(u, v)$. The foveated area is generated by modulating the value of $\tau$ using an exponential function around the focal point (Figure 4 (middle)). Changing the variance $\sigma$, changes the size of the foveated area. The density of the foveated area is denoted by $P_f(u, v)$.

$$P_f(u, v) = e^{-0.5\left(f_x^2 + f_y^2\right)\sigma} \tag{1}$$

where $f_x$ and $f_y$ are the current focal position. Combining both components, we calculate $\tau$ as follows.

$$\tau(u, v) = (1 - P_b(u, v)) \cdot P_f(u, v) + P_b(u, v) \tag{2}$$

With the resulting sampling mask, the volume is selectively rendered at all positions $(u, v)$ where $M(u, v) = 1$ (Figure 4 (right)).

This process is repeated for every new frame that is rendered. To guarantee uniform sampling and maximize the amount of visual information that can be accumulated over time, the underlying blue noise maps are changed every time. Due to the computational complexity of blue noise generation, we use a pre-calculated series of 64 noise tiles. We loop the series to render frame sequences of arbitrary length.

Sampling only a small subset of rays can drastically reduce the computation time per frame. We define the maximum possible compression rate $C_{max}$ of the technique as follows.

$$C_{max} = \frac{1}{h \cdot w} \sum_{u=0, v=0}^{h, w} \tau(u, v) \tag{3}$$

where $h$ and $w$ are the spatial dimensions of the framebuffer. $P_b$ and $P_f$ are the probability of casting a ray at $u$, $v$ as described above.

**3.1.3 Rendering—**In a naive implementation of the renderer in OptiX and CUDA, invalid pixels would simply be discarded on the kernel level. However, using one kernel thread per full-size framebuffer pixel will yield only negligible performance gains. This is because GPU kernel calls are grouped in warps. Results are available only after all threads in a warp conclude. We develop two methods to circumvent this issue.

**3.1.4 Direct Sampling—**For this method, we create a stochastic function $P$ that incorporates both $P_b$ and $P_f$. It adapts over time as the noise pattern and the location of the fovea change. The function can be called to generate a position $u$, $v$ within the bounds of the framebuffer. Specific values for $u$ and $v$ are dependent on both probability functions. Therefore, it is more likely to generate a position in and around the foveated area. When rendering a new frame, we allocate a small framebuffer in which the number of pixels corresponds to the desired sampling compression $C_{max}$ which means that each kernel thread contributes calculations without potentially being discarded. For each entry in this buffer, the renderer queries $P$ to determine which pixel to render to the compact frame buffer. The values for $(u, v)$ relate to a pixel's position in the full-size framebuffer. Therefore, the result is a compact representation of the full frame. The contents of this compact buffer are projected back to their respective positions in the full-size framebuffer to generate the sparse image. Figure 5 visualizes this process.

Direct sampling is relatively unintrusive in terms of pipeline integration. Instead of issuing a CUDA kernel run across the full image dimensions, we call it on a smaller buffer. Function $P$ acts as a proxy when accessing the screen-space coordinates in each GPU thread. The downside of this method is that $P$ does not reliably produce unique sampling positions. Although direct sampling performs better than the naive approach, we encounter duplicate coordinates quite frequently, making the renderer recompute existing samples, forfeiting potential compression.

**3.1.5 Stream Compaction—**To alleviate the problems with direct sampling, we separate the sample map generation from the actual sampling. In the first step, the sampling map is generated using the approach described in Section 3.1.2. Next, a stream compaction algorithm is used to remove sparsity in the sampling map (Figure 6). This allows us to pack all valid pixels in the map into a smaller framebuffer similar to the direct sampling approach.

The rendering is performed on this smaller framebuffer, and the same back-projection mechanism restores the full frame image (Figure 6).

The mask generation and compaction steps are implemented in CUDA to accelerate the process. Using this approach, we effectively circumvent the problems encountered in direct sampling and are therefore able to further approach the theoretical $C_{max}$.

## 3.2 Reconstruction Network

The core of FoVolNet is a deep neural network. We have developed a two-stage hybrid architecture that is based on the W-Net architecture [55]. It draws ideas from both direct prediction and kernel prediction approaches. We specifically design this network to accommodate sparse frames with minimal input features. Images are reconstructed solely from RGB input—optical flow or other auxiliary features are not required. The network's components are described in detail here.

**3.2.1 Overall Design—**The core idea of our design is to split the reconstruction process into two steps. The split is realized by running two networks in sequence (Figure 7).

This hybrid architecture employs both direct and kernel prediction. Without the initial reconstruction step, the kernel prediction method fails to perform due to the absence of rich pixel neighborhoods from which to draw visual information. On the other hand, performing only the direct prediction step would result in sub-par image quality, as we show in the evaluation.

**3.2.2 Direct Prediction: Coarse Image Reconstruction—**Network $D$ (Figure 7) reconstructs the image using direct prediction. That is, its output $O_d$ is directly interpretable as an image. This step reconstructs the overall features of the frame and fills the blank spots between valid pixels in the input. In addition to this, we preserve the decoder's hidden state $H_d$ for further processing.

**3.2.3 Kernel Prediction: Image Refinement—**In the second reconstruction step, network $K$ predicts convolutional kernels on multiple scales in both encoder and decoder stages. They are then applied in sequence to $O_d$. The kernel prediction stage takes the hidden states $H_d$ of $D's$ decoder stage (Figure 7) and forwards them to $K's$ convolution blocks in both the encoder and decoder stages. The convolution blocks are used to predict filter kernels from $H_d$. Network $K's$ input image is passed through the network by applying each block's predicted filter to the image in sequence. This process is analogous to the original W-Net filtering approach [55].

This step allows us to remove any remaining artifacts and blurriness that might result from direct prediction. Using the pre-filtered output $O_d$, we provide visual context for the filters to refine the image meaningfully. When using the original sparse image as input for this stage, we saw blotchy artifacts in areas with insufficient visual information to cleanly filter the image, and more densely sampled areas generally looked blurrier.

#### 3.2.4 Recurrence—

We introduce recurrent connections in multiple parts of the network to accumulate state. The aggregated information aids the reconstruction of temporally stable image sequences.

Decoder blocks in network $D$ are connected by recurrent connections that pass down the block's output hidden state back to its input in the next training step. On a broader scale, the output $O_d$ of network $D$ is passed back to its input layer as part of the data of the subsequent run. Current and recurrent states are combined using a concatenation operation along the channel dimension, and appropriate up- or down-sampling is applied to make all inputs compatible for subsequent operations.

### 3.3 Loss

Model optimization was performed using a linear combination of multiple loss components. We split the training loss into a spatial and a temporal component which we label $L_s$ and $L_t$, respectively. The losses are defined as follows.

$$L = \lambda_s L_s + \lambda_t L_t \tag{4}$$

where $\lambda_{s,t}$ denote the linear weights assigned to the loss. In our training we choose $\lambda_s = 0.8$ and $\lambda_t = 1.0$ which—given the losses' different magnitudes—weights spatial and temporal components at a ratio of 10:1. This balance of image quality and temporal stability was ideal for our trainings but might vary per training dataset. For $L_s$ we use a combination of $L_1$ and VGG19-based LPIPS perceptual loss [65] terms (Figure 8). The temporal loss is a combination of $L_1$ loss and optical flow (OF) loss as used by Kaplanyan et al. [21].

$$L_s = \lambda_1 LPIPS + \lambda_2 L_1 \quad \text{and} \quad L_t = \lambda_3 L_1 + \lambda_4 OF \tag{5}$$

We choose $\lambda_1 = 0.9$, $\lambda_2 = 0.1$, $\lambda_3 = 1.0$, and $\lambda_4 = 0.1$ for the linear weights to equalize the components' magnitudes. Overall, the perceptual loss alone provides good reconstruction quality; however, adding a small $L_1$ term helps preserve some more high-frequency details.

#### 3.3.1 Spatial Loss—

During training, both $L_s$ and $L_t$ are applied across the whole series of images in a sequence. Both losses are computed for each time step and the model weights are updated once after a full sequence of loss values has accumulated. Given any pair of predictions $y_p$ and ground truth targets $y_g$ with $t$ total time steps, we calculate $L_s$ as follows.

$$L_s(y_g, y_p) = \sum_{i=0}^{t} \left(1 - e^{-0.5i}\right) \cdot \left(\lambda_1 LPIPS\left(y_{g_i}, y_{p_i}\right) + \lambda_2 \left\| y_{p_i} - y_{g_i} \right\|_1\right) \tag{6}$$

Early images in the sequence are exponentially down-weighted to account for errors due to the lack of recurrent state at the beginning.

**3.3.2 Temporal Loss—**Using $L_s$ alone provides good single frame reconstruction quality. However, as previous studies [4, 15, 56] have noted, it results in temporal flickering. Hasselgren et al. [15] have shown that adding a simple $L_1$ term helps reduce temporal flickering drastically, but we have found their method to be prone to tearing artifacts when there is fast movement between adjacent frames. We add a small optical flow term, as used by Kaplanyan et al. [21] to stabilize such cases. The first component forces the network to produce adjacent frames with finite differences similar to the output. The optical flow loss works by comparing the current frame against its predecessor. The previous frame is warped using the optical flow $\phi_{(i-1)\to i}$ with the warping operator $\omega$ to match the current frame. The network has to match this warped frame, leading to less tearing in the final output as consecutive frames become similar to their respective predecessors. For a sequence of $t$ images $L_t$ is defined as follows.

$$L_t(y_g, y_p) = \sum_{i=1}^{t}\sum_{j=0}^{i-1} \lambda_3 \left\| \left(y_{p_i} - y_{p_j}\right) - \left(y_{g_i} - y_{g_j}\right) \right\|_1 + \sum_{i=1}^{t} \lambda_4 \left\| y_{p_i} - \omega\left(y_{p_{i-1}}, \phi_{(i-1)\to i}\right) \right\|_1 \tag{7}$$

In our training, the first loss term is defined over the whole sequence of prior frames. This way of constructing the loss emphasizes later image pairs in the sequence which entices training to use recurrent connections. On the other hand, the optical flow loss is only applied to a frame’s direct predecessor as warping frames becomes harder and more prone to errors the farther they are apart temporally. We use Hanika et al.’s fast reprojection algorithm [14] to estimate optical flow between frames during training. For both components of $L_t$, we do not consider the first frame of the sequence as it has no viable predecessor.

## 3.4 Model Precision & Optimization

Initially, we train the network using full 32-bit floating-point precision. However, the computational cost of running a full-precision network is often unnecessarily high. We truncate the network’s weights to 16-bit half precision format as a first optimization step. This operation does not cause any noticeable performance loss.

We also experiment with post-training quantization on a pre-trained model. In this process, we initially train the model in full-precision mode. After this, the precision of the network is reduced, and training continues using half-precision. In contrast to similar works [20, 55], we do not simulate integer quantization [26], as we observed drastic deterioration of image quality using this approach on our data. This is likely due to the sparsity of the input and the reliance on network $D$ to contribute to the final output instead of just extracting features.

Post-training quantization is realized using TensorRT [38]. Models are trained in PyTorch [8] and exported to ONNX format. We then transform the ONNX network to a CUDA inference engine using the tools provided by TensorRT [38]. In our trials, we experiment with different levels of optimization. Namely, we choose from different numerical precisions: float32, float16, int8, and mixed-mode. We compare different settings in Section 4.

### 3.5 Training

We provide information about network configuration, hyperparameters, and details regarding the dataset used for training in this section. The training was performed on two NVIDIA Quadro RTX 8000 GPUs.

**3.5.1 Model Configuration—**Both sub-networks $D$ and $K$ of our reconstruction network are based on the U-Net design [43]. Each network has four encoder and three decoder blocks. Skip connections connect the blocks. In network W:\Production\18192\413826MC\0001\Graphics, each block consists of two convolutional layers of equal depth, followed by a ReLU activation. On the other hand, each block in network W:\Production\18192\413826MC\0001\Graphics only has a single convolution to predict the kernels. Both networks follow the same progression of block configurations which is defined as:

$$e64 - e64 - e80 - d96 - d80 - d64 - d64 \tag{8}$$

where $e$ and $d$ denote encoder/decoder blocks followed by the convolution depth used for *Conv2D* layers in the block. Blocks in the encoder stage conclude with an average pooling layer to downscale the image. Analogously, all except the final block in the decoder part up-sample their outputs.

**3.5.2 Dataset—**FoVolNet is trained on short video sequences of several pre-rendered volume datasets. The data covers CT scans of humans, animals, mechanical parts, and large-scale simulation data from astronomy and material sciences (Table 1).

We render the datasets at a resolution of 800×800 using the previously described renderer. Video sequences consist of a continuous camera fly-through around the volume to cover most angles of the data. For training, the video dataset is sliced into 16-frame segments with no overlaps, and images are tiled at a resolution of 256×256. We find that these spatiotemporal dimensions offer the best trade-off between training time and quality. Sequences with eight or fewer frames resulted in under-utilization of recurrent connections and, therefore, bad temporal coherence. Batches consist of 15 such 16×256×256 sequences. In total, the network is trained on 16000 unique images. Our validation data consists of 1600 unique images from the same datasets. Training, validation, and test datasets were split randomly at a 10:1:1 ratio.

**3.5.3 Data Augmentation—**We use data augmentation, which effectively increases the number of unique training sequences we provide to the network. During training, sequences of frames are subject to random augmentations to improve training effectiveness. Table 2 shows the list of augmentations used during training. Here, $\mathrm{P}(\mathrm{x})$ denotes the independent probability of each augmentation occurring for any given batch of data.

**3.5.4 Hyperparameters—**During development, we experimented with different sets of hyperparameters to empirically determine the best settings for training. These include initial learning rate (LR), learning rate schedule, optimizer, weight decay, and length of training. For the final version, we use the following setup.

The LR is set to an initial value of $1.25e-3$, and a cosine annealing LR schedule is applied to gradually reduce the LR to a minimum value of $1e-8$. We use the Ranger [63] optimizer with a weight decay set to $1e-2$ to stabilize training. The model usually converges at around epoch 60–80. All trainings are stopped after 120 epochs.

## 4 Evaluation

To show the potential of FoVolNet, we conduct several tests to evaluate different aspects of the system. All evaluations are conducted using our custom foveated rendering pipeline.

### 4.1 System Setup

We use C++ backends for both PyTorch [8] and TensorRT [38] for inference. All evaluations were performed on an end-user machine with an Intel Core i7–6900K CPU with 128 gigabytes of RAM and an NVIDIA Titan RTX GPU. The system runs Ubuntu 20.04 LTS, and all parts of FoVolNet were developed and compiled on Linux. All frames are rendered at a resolution of 1280 × 720. We use our ray marching renderer and enable global shadows using a single light source per scene. Layer weights are truncated to fp16 precision, unless otherwise specified. All output was produced using images that were not in the training set.

Some results show comparisons to DeepFovea—the current state-of-the-art of foveated sparse frame reconstruction [21]. We train a model of this architecture on our data in our training pipeline using hyperparameters as suggested by the authors.

### 4.2 Inference Speed & Pipeline Throughput

To test the rendering throughput of our system, we use a fixed-path camera fly-through in our rendering pipeline. The camera path consists of a pattern of oscillating zoom with continuous rotation in two axes. This allows us to cover most aspects of the data using a small number of frames. Please refer to the supplemental video for more details.

A comparison of conventional raymarching with FoVolNet is shown in Figure 9. We perform the fly-through mentioned above on the Vortices 2 dataset. Baseline rendering time is drastically reduced due to sparse, foveated sampling of the volume. A constant, scene-independent inference time adds to the total frame time. The result is a sequence of fast and stable frame times that is less dependent on camera angle or scene configuration. As the thumbnails in Figure 9 suggest, the more screen space is occupied by data, the larger the benefit of using our technique. However, as the left-most image shows, we achieve roughly two times faster end-to-end rendering performance even from a far-away viewing position.

A more comprehensive analysis of rendering performance is shown in Figure 10. Here, we compare against DeepFovea [21]. The sequences were created at two different quality settings, which differ in their configuration of sampling density. The hatched parts of the deep learning based runs indicate the inference times. We report the resulting average end-to-end speedups for all datasets in Table 3. Note that the fly-through sequences are all composed of roughly equal parts far-away and close-up viewing positions. This is due to the oscillating zoom of the camera. Therefore, our results represent speedups that can be expected in the average case. However, if the data is viewed at a reasonably close angle like shown in Figure 1, speedups are generally much higher.

### 4.3 Image Quality

Final image quality is at least as important as inference speed when it comes to image reconstruction. For all datasets, we calculate structural similarity (SSIM) and peak signal to noise ratio (PSNR) on single frames and image sequences. The image matrix in Figure 11 shows results for both foveated areas and the periphery on single frames. Our architecture can reconstruct fine details even in peripheral areas of the frame. Notice how pure direct prediction methods like the recurrent U-Net used in DeepFovea [21] fail to preserve high-frequency details as the number of samples decreases.

For the video analysis, we create a camera fly-through video of the CHAMELEON dataset. We split the video into multiple 50-frame sequences (of which we show two) with a jump-cut between them. Figures 12 and 13 show the reconstruction quality over time. A red line indicates the cut. Both FoVolNet and direct prediction improve their quality over a short ramp-up period in which state is accumulated. Similarly, the network needs several frames after the jump cut to recover full quality. However, the hybrid architecture outperforms direct prediction by a constant offset.

### 4.4 Temporal Stability

The quality of temporal coherence is evaluated on the same fly-through clips. The sequences were constructed so that they each start and end in fast camera movement while slowing down towards the middle. This three-act setup allows us to see how the network uses accumulated state to retain temporal consistency when there is (1) plenty of movement with little prior state, (2) little movement but lots of state, and lastly, (3) lots of movement and lots of state. We compute the temporal PSNR (tPSNR) as used by Hasselgren et al. [15]. This value is the PSNR of finite differences between frames. Instead of computing the PSNR on the image itself, we compute it on the difference between the current and previous frame. The aforementioned fast-slow-fast pattern is reflected by data in Figure 14. It shows that FoVolNet is able to retain stability throughout most of the sequences. Both models achieve peak quality when there is little movement. This is unsurprising since, without movement, the network acts as a simple accumulation buffer. However, FoVolNet is able to retain quality under much faster movement than direct prediction—especially in Phase (3) when there is enough state available to the layers. Please refer to the supplemental material for the source clips.

In addition to this metric, we provide just-objectionable-difference (JOD) [42] scores for the whole video. This score is similar to just-noticeable-difference (JND), but instead of quantifying the difference between pairs of images, it is better suited to compare multiple degraded images to the reference. That means that while the results of different reconstruction methods might look degraded in different ways, they will still have similar JOD scores as they are equally different from the ground truth. The data was produced using FovVideoVDP [35]. We use the default settings for a 4K screen viewed under office light levels. Detailed settings were chosen as follows: 75.4 [pix/deg], Lpeak=200, Lblack=0.5979 [cd/m$^2$], non-foveated, (standard_4k). Data is produced for the video at different quality settings as shown in Table 4. Note that higher scores are better, with 10 being the maximum score.

### 4.5 Model Precision

During training, we configure the weights to use full 32-bit precision. By default, this level of precision is retained during inference. However, reducing the precision of certain weights in the network can drastically improve the performance during inference. We examine the effects that such adjustments have on image quality in practice (Figure 15).

In our tests, quantization artifacts were most apparent on homogeneous surfaces, subtle gradients, and transparent regions. Two examples are shown in Figure 15. The difference in quality is especially apparent in the lower dataset shown in Figure 15. The fading color towards the top shows a much more abrupt cut-off in quantized precisions (int8 and mixed-precision int8/fp16) compared to their unquantized counterparts (fp32 and fp16). The brightness of the lower frame was increased to emphasize the subtle differences. There was no noticeable difference between the non-quantized precisions fp32 and fp16.

### 4.6 Effective Compression Rate

Reducing the number of total pixels that the volume needs to be sampled at immediately affects rendering performance. In the ideal case, a sparse rendering algorithm would achieve rendering times that scale linearly with the number of pixels. We termed this ideal case $C_{max}$ —the maximum possible rendering performance at any given sparsity level. To test how well our stream compaction sparse renderer performs, we record frame times along the whole spectrum of sparsity as represented by $\tau$ ranging from 1.0 (full frame) to 0.0 (no samples). Data is recorded for both the stream compaction method and a naive rejection approach which simply skips computations for certain threads on a full-frame kernel run. The results of this test are shown in Figure 16.

The data shows that our compaction method maps well to $C_{max}$. As $\tau$ approaches sparsity rates of around 10%, the performance starts to diverge slightly from $C_{max}$. Due to hardware limitations and computational overhead in the pipeline, the curve starts to flatten at around 1%. In our experiments, values for $\tau$ reside in the range of 0.1 to 0.01, which equates to roughly 10× to 25× compression rates compared to the naive approach, which is almost ten times less efficient.

## 5 Limitations & Future Directions

FoVolNet works well on various data, as we have shown in the evaluation. It can benefit from more specialized training to address some of the edge cases we encountered during development, like high-frequency visual content or more pronounced transparency. Beyond this, there are several interesting extensions that we suggest here.

### High-frequency Content.

Regions of a volume that contained high-frequency intensity shifts resulted in increased temporal flickering when using our network. In most cases, increasing the renderer's volume sampling rate would alleviate such issues. A more cost-effective approach is to purposely introduce such artifacts into the training data, which would reduce the overall severity of the issue. Another approach is to emphasize temporal loss terms by increasing their weight (sacrificing visual quality) or by introducing more adaptable terms like a GAN critique [11].

### Beyond Raymarching.

In this work, we showcase our technique on the example of a raymarching renderer. However, FoVolNet is easy to extend to Monte Carlo methods like volume path tracing. Here we see potential to stabilize framerates by cutting short long-running threads due to multiple bounces and reconstructing their results using constant-time neural networks. Support for other data types like particle volumes or flow fields could also be added. We encourage further research to explore the specifics of such extensions.

### Neural Adaptive Sampling.

The approach presented here can be improved by predicting adaptive sample maps. Similar to Stengel et al.'s approach [52], both adaptive and foveated maps can be merged to maximize visual quality. Creating more off-focus sampling density could also improve the remaining issues with temporal stability. This extension increases sampling efficiency by replacing the naive uniform sampling in the periphery with an overall smarter approach.

### Beyond the Screen.

High-fidelity immersive visualization of volume data is still out of reach today. However, with FoVolNet we achieve higher and more consistent framerates (Figure 9). With further improvements to the network, this goal could be attained much sooner than with conventional rendering techniques. The inference overhead could be reduced to a point where real-time, high-fidelity rendering becomes possible. This would open up opportunities to utilize FoVolNet for immersive experiences of volume data in VR.

## 6 Conclusion

We presented FoVolNet—a foveated neural reconstruction system for volume visualization. FoVolNet accelerates conventional volume rendering techniques by sparsely sampling the data and reconstructing the full-frame using deep learning. Our novel network design reconstructs the final rendering at high quality using a hybrid of direct and kernel prediction mechanisms. We show that FoVolNet is able to provide tremendous speed-ups at compression rates as high as 25× over the state-of-the-art control technique DeepFovea [21] while preserving image quality close to the original. Our uncomplicated design makes it easy to be integrated into existing rendering pipelines.

It is our plan to combine this technique with neural representation compression techniques and streaming technology to push the field further towards real-time high-fidelity volume visualization on consumer hardware. There are numerous opportunities to use and extend this technique, and we hope to entice the visualization community to take up this pursuit.

## Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

## Acknowledgments

This research is sponsored in part by the Department of Energy through grant SC-DE0019486 and Intel’s oneAPI Centers of Excellence grant.

## References

[1]. Adler F Adler’s Physiology of the Eye. Elsevier Health Sciences, 2011.

[2]. Bako S, Vogels T, McWilliams B, Meyer M, Novák J, Harvill A, Sen P, Derose T, and Rousselle F Kernel-predicting convolutional networks for denoising monte carlo renderings. ACM Transactions on Graphics, 36(4):97:1–97:14, 2017. doi: 10.1145/3072959.3073708

[3]. Beyer J, Hadwiger M, and Pfister H A survey of GPU-based large-scale volume visualization. In Borgo R, Maciejewski R, and Viola I, eds., EuroVis—STARs, pp. 105–123. The Eurographics Association, 2014. doi: 10.2312/eurovisstar.20141175

[4]. Chaitanya CRA, Kaplanyan AS, Schied C, Salvi M, Lefohn A, Nowrouzezahrai D, and Aila T Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. ACM Transactions on Graphics, 36(4):98:1–98:12, 2017. doi: 10.1145/3072959.3073601

[5]. Dappa E, Higashigaito K, Fornaro J, Leschka S, Wildermuth S, and Alkadhi H Cinematic rendering–an alternative to volume rendering for 3d computed tomography imaging. Insights Into Imaging, 7(6):849–856, 2016. doi: 10.1007/s13244-016-0518-1 [PubMed: 27628743]

[6]. Díaz J, Vázquez P-P, Navazo I, and Duguet F Real-time ambient occlusion and halos with summed area tables. Computers & Graphics, 34(4):337–350, 2010. doi: 10.1016/j.cag.2010.03.005

[7]. Engel D and Ropinski T Deep volumetric ambient occlusion. IEEE Transactions on Visualization and Computer Graphics, 27(2):1268–1278, 2020. doi: 10.1109/TVCG.2020.3030344

[8]. Facebook Inc. PyTorch. https://pytorch.org/, 2022. [Online; accessed 02-March-2022].

[9]. Farnebäck G Two-frame motion estimation based on polynomial expansion. In Bigun J and Gustavsson T, eds., Image Analysis, pp. 363–370. Springer, Berlin, Heidelberg, 2003. doi: 10.1007/3-540-45103-X_50

[10]. Gharbi M, Li TM, Aittala M, Lehtinen J, and Durand F Sample-based monte carlo denoising using a kernel-splatting network. ACM Transactions on Graphics, 38(4):125:1–125:12, 2019. doi: 10.1145/3306346.3322954

[11]. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, and Bengio Y Generative adversarial networks. Commun. ACM, 63(11):139–144, oct 2020. doi: 10.1145/3422622

[12]. Guenter B, Finch M, Drucker S, Tan D, and Snyder J Foveated 3d graphics. ACM Transactions on Graphics, 31(6):164:1–164:10, nov 2012. doi: 10.1145/2366145.2366183

[13]. Hadwiger M, Beyer J, Jeong W-K, and Pfister H Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. IEEE Transactions on Visualization and Computer Graphics, 18(12):2285–2294, 2012. doi: 10.1109/TVCG.2012.240 [PubMed: 26357136]

[14]. Hanika J, Tessari L, and Dachsbacher C Fast temporal reprojection without motion vectors. Journal of Computer Graphics Techniques (JCGT), 10(3):19–45, 2021.

[15]. Hasselgren J, Munkberg J, Salvi M, Patney A, and Lefohn A Neural temporal adaptive sampling and denoising. Computer Graphics Forum, 39(2):147–155, 2020. doi: 10.1111/cgf.13919

[16]. Hernell F, Ljung P, and Ynnerman A Local ambient occlusion in direct volume rendering. IEEE Transactions on Visualization and Computer Graphics, 16(4):548–559, 2009. doi: 10.1109/TVCG.2009.45

[17]. Hofmann N, Martschinke J, Engel K, and Stamminger M Neural denoising for path tracing of medical volumetric data. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 3(2):13:1–13:18, 2020. doi: 10.1145/3406181

[18]. Huo Y and eui Yoon S A survey on deep learning-based monte carlo denoising. Computational Visual Media, 7(2):169–185, 2021. doi: 10.1007/s41095-021-0209-9

[19]. Igouchkine O, Zhang Y, and Ma K-L Multi-material volume rendering with a physically-based surface reflection model. IEEE Transactions on Visualization and Computer Graphics, 24(12):3147–3159, 2017. doi: 10.1109/TVCG.2017.2784830 [PubMed: 29990043]

[20]. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, and Kalenichenko D Quantization and training of neural networks for efficient integer-arithmetic-only inference. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2704–2713, 2018. doi: 10.1109/CVPR.2018.00286

[21]. Kaplanyan AS, Sochenov A, Leimkühler T, Okunev M, Goodall T, and Rufo G DeepFovea: neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. ACM Transactions on Graphics, 38(6):212:1–212:13, 2019. doi: 10.1145/3355089.3356557

[22]. Kettunen M, Härkönen E, and Lehtinen J Deep convolutional reconstruction for gradient-domain rendering. ACM Transactions on Graphics, 38(4):126:1–126:12, 2019. doi: 10.1145/3306346.3323038

[23]. Klacansky P Open scientific visualization datasets. https://klacansky.com/open-scivis-datasets/.

[24]. Kniss J, Premoze S, Hansen C, and Ebert D Interactive translucent volume rendering and procedural modeling. In IEEE Visualization, pp. 109–116. IEEE, 2002. doi: 10.1109/VISUAL.2002.1183764

[25]. Kniss J, Premoze S, Hansen C, Shirley P, and McPherson A A model for volume lighting and modeling. IEEE Transactions on Visualization and Computer Graphics, 9(2):150–162, 2003. doi: 10.1109/TVCG.2003.1196003

[26]. Krishnamoorthi R Quantizing deep convolutional networks for efficient inference: a whitepaper. arXiv preprint arXiv:1806.08342, 2018. doi: 10.48550/arXiv.1806.08342

[27]. Kroes T, Post FH, and Botha CP Exposure render: an interactive photo-realistic volume rendering framework. PLoS ONE, 7(7):e38586, 2012. doi: 10.1371/journal.pone.0038586 [PubMed: 22768292]

[28]. Kroes T, Schut D, and Eisemann E Smooth probabilistic ambient occlusion for volume rendering. GPU Pro 6: Advanced Rendering Techniques, p. 475, 2015. doi: 10.1201/9781351052108-17

[29]. Le Besnerais G and Champagnat F Dense optical flow by iterative local window registration. In IEEE International Conference on Image Processing, vol. 1, pp. I–137, 2005. doi: 10.1109/ICIP.2005.1529706

[30]. Levoy M The Stanford volume data archive. https://graphics.stanford.edu/data/voldata/.

[31]. Liu N, Zhu D, Wang Z, Wei Y, and Shi M Progressive light volume for interactive volumetric illumination. Computer Animation and Virtual Worlds, 27(3–4):394–404, 2016. doi: 10.1002/cav.1706

[32]. Lu YF, Xie N, and Shen HT DMCR-GAN: adversarial denoising for monte carlo renderings with residual attention networks and hierarchical features modulation of auxiliary buffers. SIGGRAPH Asia Technical Communications, pp. 5:1–5:4, 2020. doi: 10.1145/3410700.3425426

[33]. Lundell F Out-of-core multi-resolution volume rendering of large data sets. Master's thesis, Linköping University, 2011.

[34]. Maisano J Chamaeleo calyptratus. Digital Morphology, 2003 (Online). http://digimorph.org/specimens/Chamaeleo_calyptratus/whole.

[35]. Mantiuk RK, Denes G, Chapiro A, Kaplanyan A, Rufo G, Bachy R, Lian T, and Patney A FovVideoVDP: a visible difference predictor for wide field-of-view video. ACM Transactions on Graphics, 40(4):49:1–49:19, 2021. doi: 10.1145/3450626.3459831

[36]. Max N Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics, 1(2):99–108, 1995. doi: 10.1109/2945.468400

[37]. Max N and Chen M Local and global illumination in the volume rendering integral. In Hagen H, ed., Scientific Visualization: Advanced Concepts, vol. 1 of Dagstuhl Follow-Ups, pp. 259–274. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2010. doi: 10.4230/DFU.SciViz.2010.259

[38]. NVIDIA. TensorRT. https://developer.nvidia.com/tensorrt, 2022. [Online; accessed 02-March-2022].

[39]. Oak Ridge National Lab. Supernova dataset. https://www.ornl.gov/.

[40]. Paladini G, Petkov K, Paulus J, and Engel K Optimization techniques for cloud based interactive volumetric monte carlo path tracing. Industrial Talk, EG/VGTC EuroVis, 2015.

[41]. Plyer A, Le Besnerais G, and Champagnat F Massively parallel Lucas Kanade optical flow for real-time video processing applications. Journal of Real-Time Image Processing, 11(4):713–730, 2016. doi: 10.1007/s11554-014-0423-0

[42]. Pérez-Ortiz M, Mikhailiuk A, Zerman E, Hulusic V, Valenzise G, and Mantiuk RK From pairwise comparisons and rating to a unified quality scale. IEEE Transactions on Image Processing, 29:1139–1151, 2020. doi: 10.1109/TIP.2019.2936103

[43]. Ronneberger O, Fischer P, and Brox T U-net: convolutional networks for biomedical image segmentation. In Navab N, Hornegger J, Wells WM, and Frangi AF, eds., Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pp. 234–241. Springer International Publishing, Cham, 2015. doi: 10.1007/978-3-319-24574-4_28

[44]. Ropinski T, Döring C, and Rezk-Salama C Interactive volumetric lighting simulating scattering and shadowing. In IEEE Pacific Visualization Symposium (PacificVis), pp. 169–176. IEEE, 2010. doi: 10.1109/PACIFICVIS.2010.5429594

[45]. Ropinski T, Meyer-Spradow J, Diepenbrock S, Mensmann J, and Hinrichs K Interactive volume rendering with dynamic ambient occlusion and color bleeding. Computer Graphics Forum, 27(2):567–576, 2008. doi: 10.1111/j.1467-8659.2008.01154.x

[46]. Ruiz M, Boada I, Viola I, Bruckner S, Feixas M, and Sbert M Obscurance-based volume rendering framework. In Hege H-C, Laid-law D, Pajarola R, and Staadt O, eds., IEEE/EG Symposium on Volume and Point-Based Graphics, pp. 113–120. The Eurographics Association, 2008. doi: 10.2312/VG/VG-PBG08/113-120

[47]. Schott M, Pegoraro V, Hansen C, Boulanger K, and Bouatouch K A directional occlusion shading model for interactive direct volume rendering. Computer Graphics Forum, 28(3):855–862, 2009. doi: 10.1111/j.1467-8659.2009.01464.x

[48]. Shih M, Rizzi S, Insley J, Uram T, Vishwanath V, Hereld M, Papka ME, and Ma K-L Parallel distributed, GPU-accelerated, advanced lighting calculations for large-scale volume visualization. In IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV), pp. 47–55. IEEE, 2016. doi: 10.1109/LDAV.2016.7874309

[49]. Shih M, Zhang Y, Ma K-L, Sitaraman J, and Mavriplis D Out-of-core visualization of time-varying hybrid-grid volume data. In IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), pp. 93–100, 2014. doi: 10.1109/LDAV.2014.7013209

[50]. Silver D Vortices volume dataset. https://mbs.rutgers.edu/staff/deborah-silver/.

[51]. Šoltészová V, Patel D, Bruckner S, and Viola I A multidirectional occlusion shading model for direct volume rendering. Computer Graphics Forum, 29(3):883–891, 2010. doi: 10.1111/j.1467-8659.2009.01695.x

[52]. Stengel M, Grogorick S, Eisemann M, and Magnor M Adaptive image-space sampling for gaze-contingent real-time rendering. Computer Graphics Forum, 35(4):129–139, 2016. doi: 10.1111/cgf.12956

[53]. Sundén E, Ynnerman A, and Ropinski T Image plane sweep volume illumination. IEEE Transactions on Visualization and Computer Graphics, 17(12):2125–2134, 2011. doi: 10.1109/TVCG.2011.211 [PubMed: 22034331]

[54]. Sánchez Pérez J, Meinhardt-Llopis E, and Facciolo G TV-L1 optical flow estimation. Image Processing On Line, 3:137–150, 2013. doi: 10.5201/ipol.2013.26

[55]. Thomas MM, Vaidyanathan K, Liktor G, and Forbes AG A reduced-precision network for image reconstruction. ACM Transactions on Graphics, 39(6):231:1–231:12, 2020. doi: 10.1145/3414685.3417786

[56]. Vogels T, Rousselle F, McWilliams B, Röthlin G, Harvill A, Adler D, Meyer M, and Novák J Denoising with kernel prediction and asymmetric loss functions. ACM Transactions on Graphics, 37(4), 2018. doi: 10.1145/3197517.3201388

[57]. Wald I, Johnson GP, Amstutz J, Brownlee C, Knoll A, Jeffers J, Günther J, and Navrátil P OSPRay—a CPU ray tracing framework for scientific visualization. IEEE Transactions on Visualization and Computer Graphics, 23(1):931–940, 2016. doi: 10.1109/TVCG.2016.2599041

[58]. Weier M, Roth T, Kruijff E, Hinkenjann A, Pérard-Gayot A, Slusallek P, and Li Y Foveated real-time ray tracing for head-mounted displays. Computer Graphics Forum, 35(7):289–298, 2016. doi: 10.1111/cgf.13026

[59]. Weier M, Stengel M, Roth T, Didyk P, Eisemann E, Eisemann M, Grogorick S, Hinkenjann A, Kruijff E, Magnor M, Myszkowski K, and Slusallek P Perception-driven accelerated rendering. Computer Graphics Forum, 36(2):611–643, 2017. doi: 10.1111/cgf.13150

[60]. Weiss S, IşIk M, Thies J, and Westermann R Learning adaptive sampling and reconstruction for volume visualization. IEEE Transactions on Visualization and Computer Graphics, 28(7):2654–2667, 2022. doi: 10.1109/TVCG.2020.3039340 [PubMed: 33211659]

[61]. Wolfe A, Morrical N, Akenine-Möller T, and Ramamoorthi R Spatiotemporal Blue Noise Masks. In Ghosh A and Wei L-Y, eds., Eurographics Symposium on Rendering, pp. 117–126. The Eurographics Association, 2022. doi: 10.2312/sr.20221161

[62]. Wong K-M and Wong T-T Robust deep residual denoising for monte carlo rendering. In SIGGRAPH Asia Technical Briefs, SA ‘18, pp. 14:1–14:4. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3283254.3283261

[63]. Wright L Ranger—a synergistic optimizer. https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer, 2019.

[64]. Xu B, Zhang J, Wang R, Xu K, Yang YL, Li C, and Tang R Adversarial monte carlo denoising with conditioned auxiliary feature modulation. ACM Transactions on Graphics, 38(6):224:1–224:12, 2019. doi: 10.1145/3355089.3356547

[65]. Zhang R, Isola P, Efros AA, Shechtman E, and Wang O The unreasonable effectiveness of deep features as a perceptual metric. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 586–595, 2018. doi: 10.1109/CVPR.2018.00068

[66]. Zhang Y and Ma K-L Fast global illumination for interactive volume visualization. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 55–62, 2013. doi: 10.1145/2448196.2448205

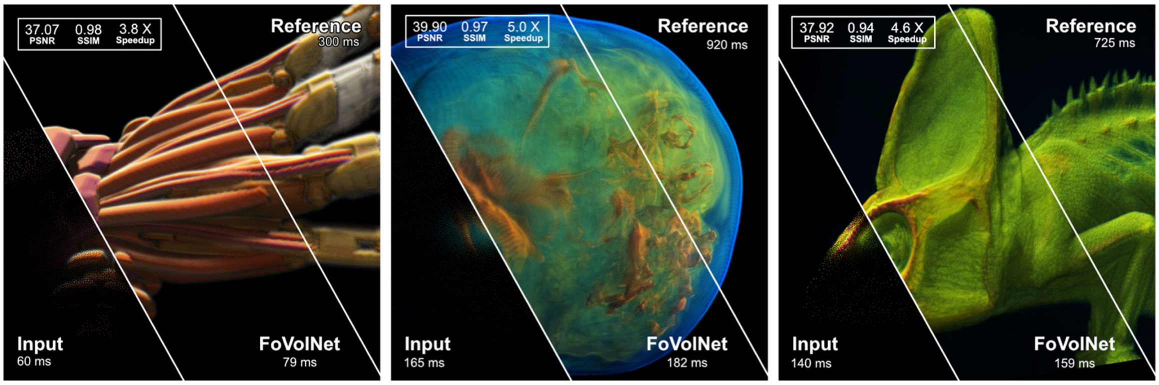

**Fig. 1:**

We propose a novel rendering pipeline for fast volume rendering using optimized foveated sparse rendering and deep neural reconstruction networks. FoVolNet can faithfully reconstruct visual information from sparse inputs. With FoVolNet, developers are able to significantly improve rendering time without sacrificing visual quality.

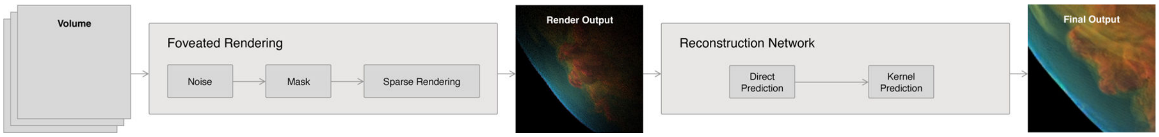

**Fig. 2:**

Overview of FoVolNet's components. The rendering pipeline loads a volume and renders it sparsely—saving time by skipping pixels in the periphery. Then, the sparse rendering is reconstructed by a neural network which takes constant time. The output of the rendering pipeline is the full-frame rendering.

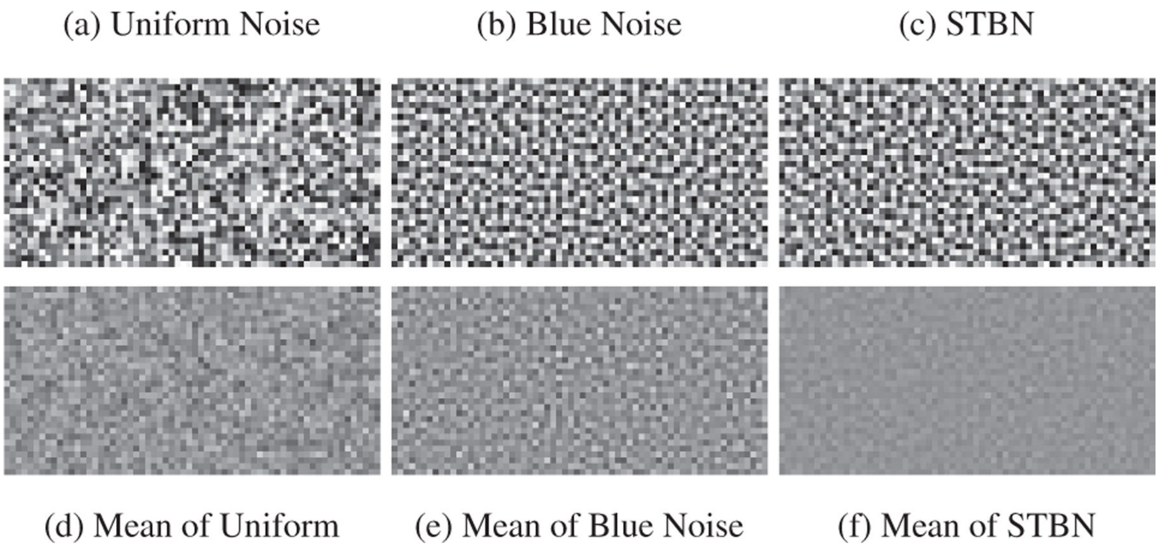

**Fig. 3:**

We compare multiple noise patterns to create our sampling masks. The top row (a)-(c) shows the different types of noise. In the bottom row we show an 8-image sequence of patterns averaged across time to emphasize their temporal stability (d)-(f).

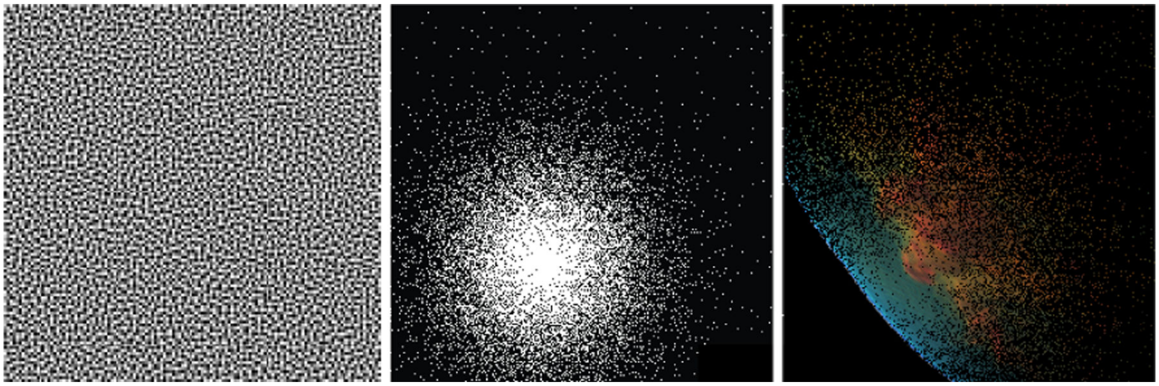

**Fig. 4:**

Sampling maps are generated using an STBN [61] (left). The area around the focal point is sampled more densely using an exponential fall-off to preserve details (middle). The volume is sparsely sampled using the sampling map (right).

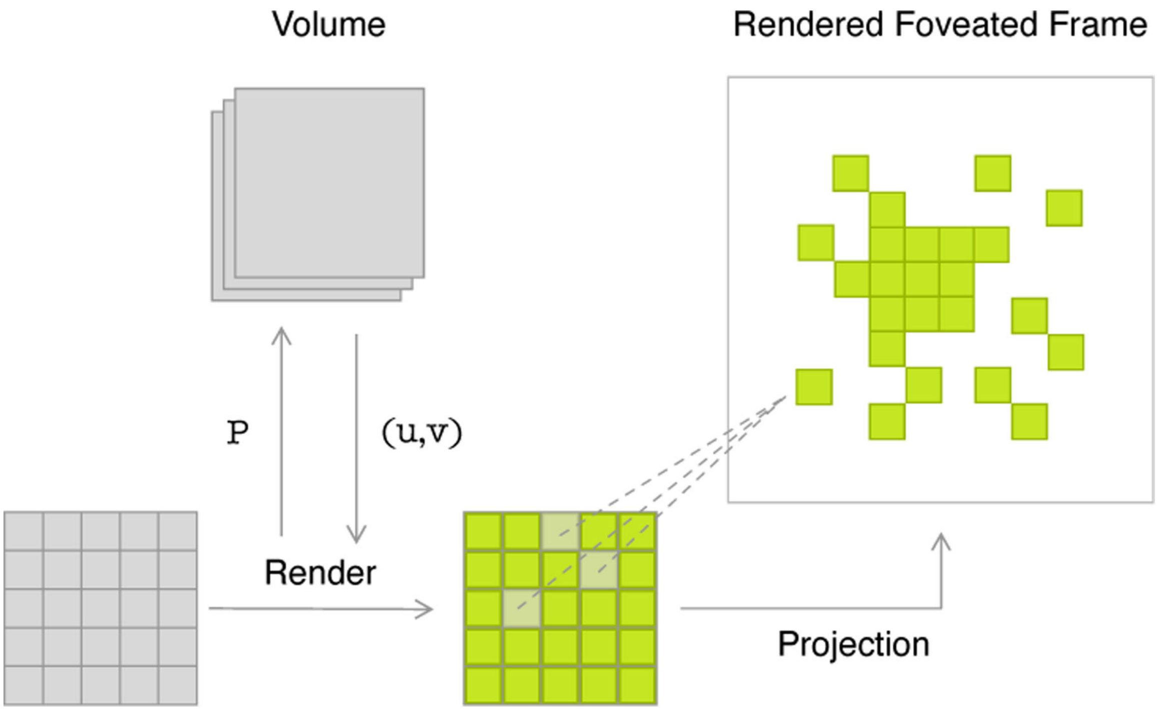

**Fig. 5:**

For direct sampling, we use a stochastic function *P* to generate sample points. A small frame buffer is filled with samples that correspond to real locations on the full-size framebuffer. Color values are reprojected to the full frame.

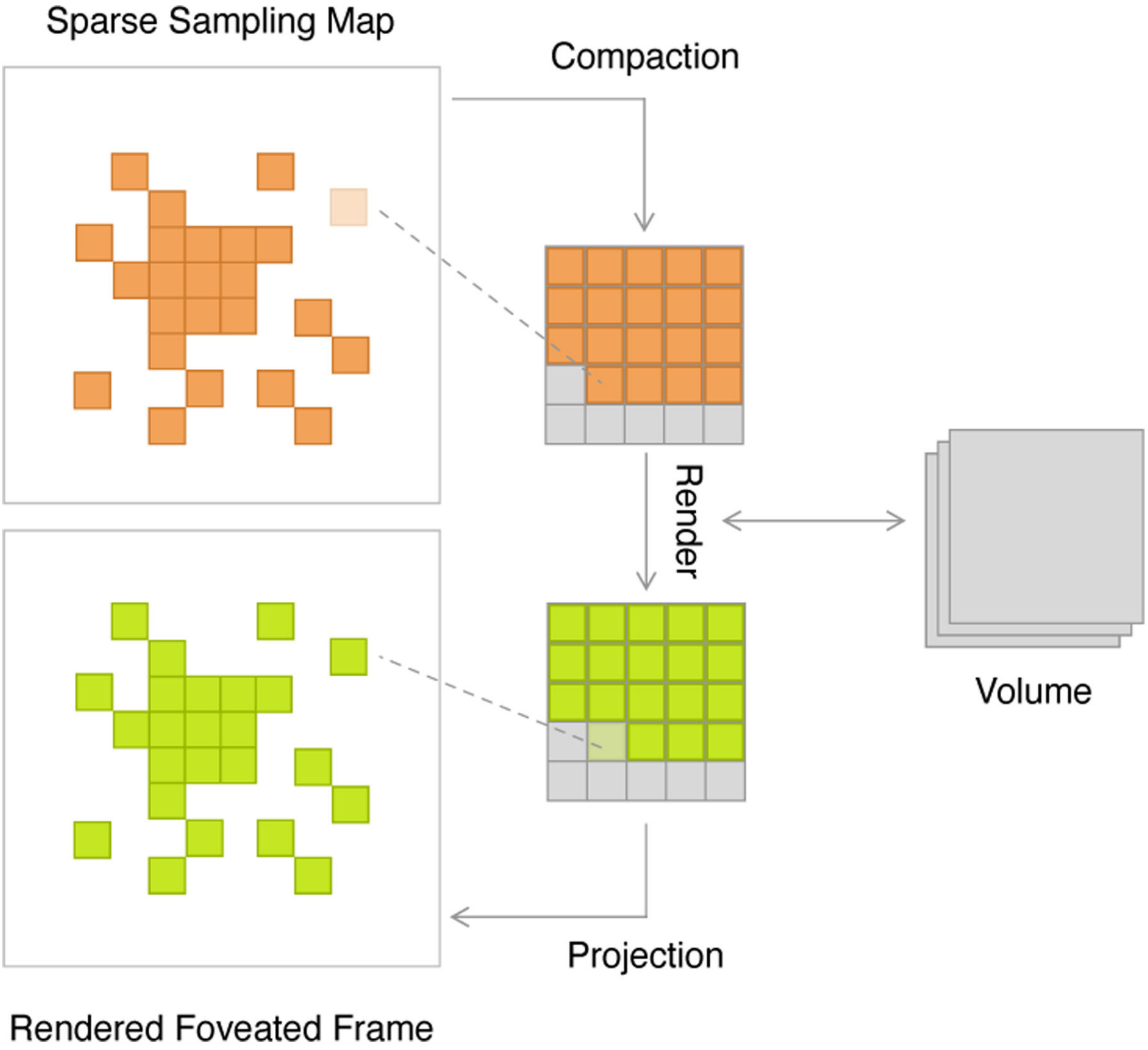

**Fig. 6:**

The renderer first creates a sampling map which is compacted into a small framebuffer. The volume is sampled according to each pixel’s ray direction in the full-size frame. After rendering, the resulting color values are projected back to the initial frame.

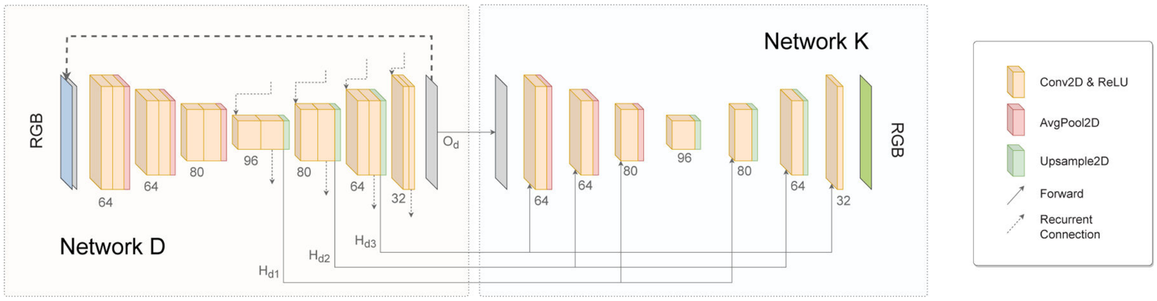

**Fig. 7:**

The reconstruction network comprises two U-Net stages $D$ and $K$. All *Conv2D* layers are configured with a stride and padding of 1 and no dilation. Network $D$ uses a 3 × 3 kernel size while $K$ uses 1 × 1 kernels. *Upsample2D* layers use bilinear interpolation for filtering. Skip connection are not shown in favor of readability. Network $D$ performs coarse reconstruction through direct prediction while the second stage $K$ uses $D$‘s hidden state to predict convolutional kernels which are subsequently applied to $D$‘s output to produce the final frame.

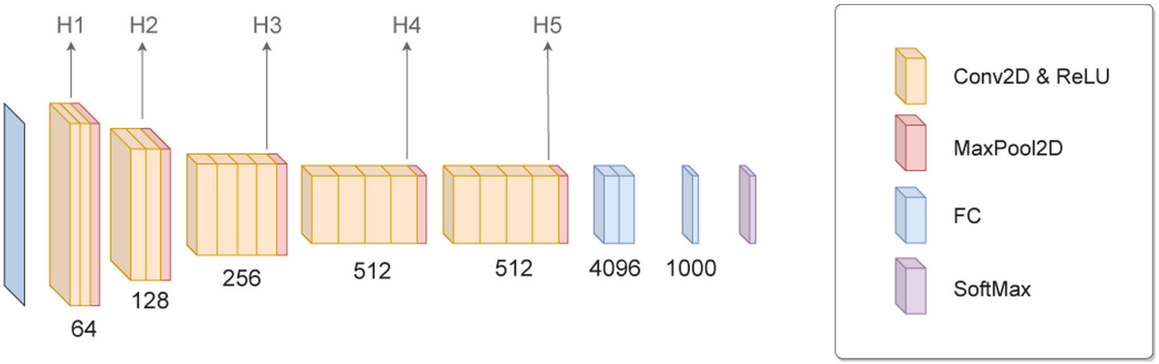

**Fig. 8:**
The hidden states H1-H5 of the VGG19 image classification network are used to measure the perceptual quality of our image reconstruction [65].

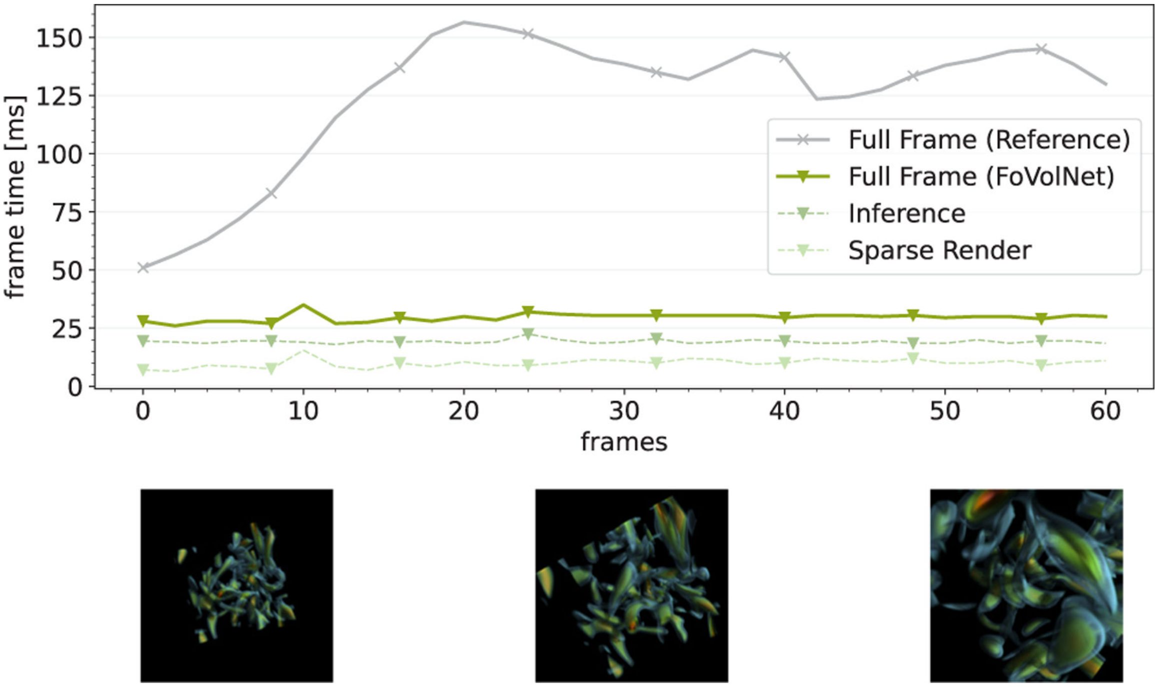

**Fig. 9:**
The per-frame timings of different pipeline components during a camera fly-through of VORTICES 2. We compare times for FoVolNet using fp16 precision with conventional DVR as the reference. Thumbnails show the camera position at that specific point in the run.

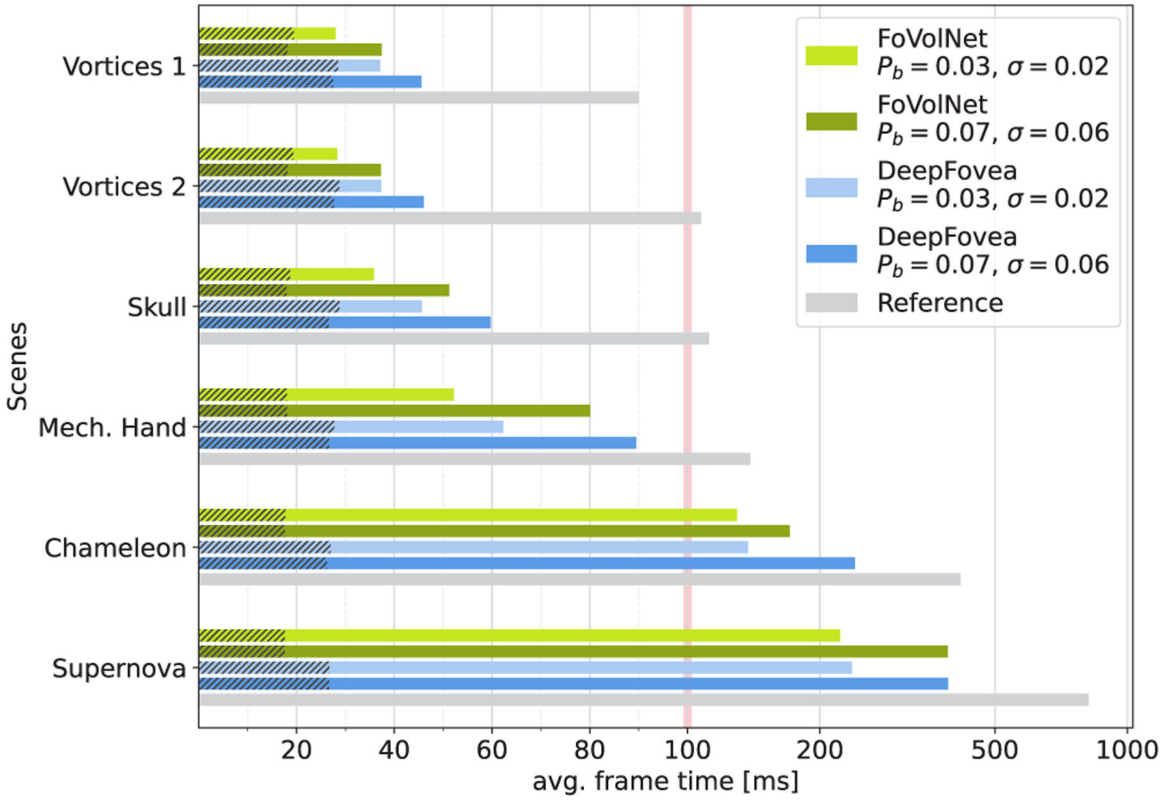

**Fig. 10:**

Average frame times from fly-through renderings of different datasets. Each dataset was rendered for 500 frames. The camera movement was framerate-independent. We compare against DeepFovea [21] as specified by the authors. Note that the x-axis scales logarithmically past frame 100 to accommodate long frame times.

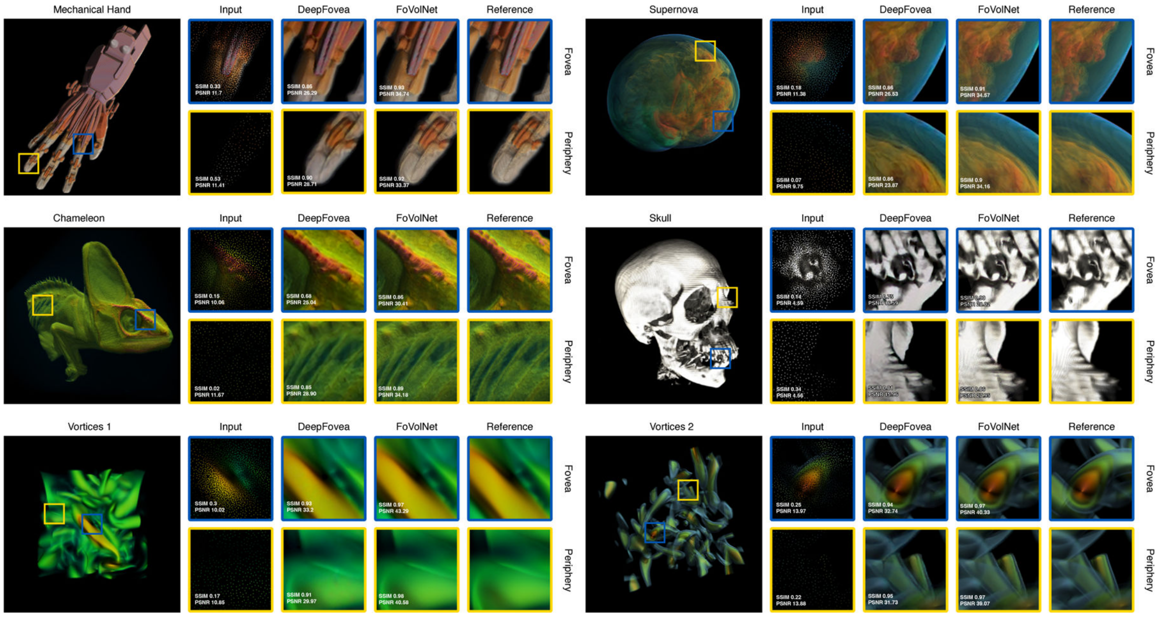

**Fig. 11:**

Visual comparison of reconstruction quality using our method. For each dataset, we show the area around the fovea in blue and a part of the periphery in yellow. All images were generated with $P_b = 0.03$ and $\sigma = 0.02$ for $P_f$.

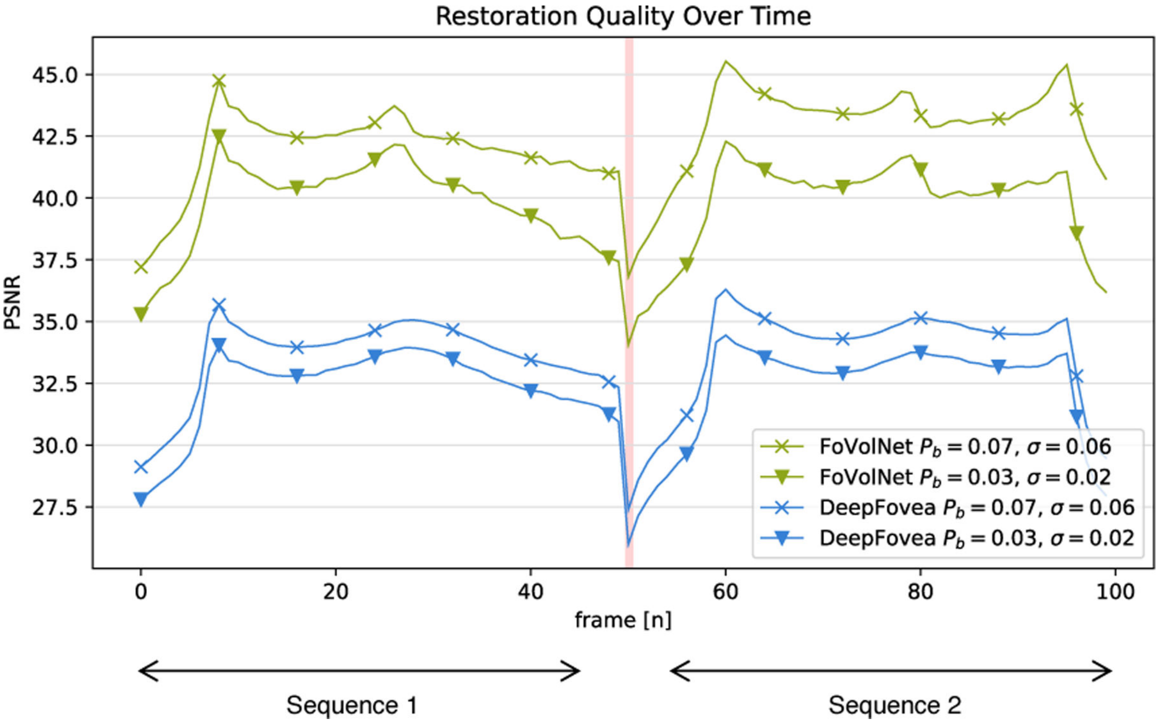

**Fig. 12:**

Still-frame PSNR over the course of two 50-frame clips from the CHAMELEON dataset.

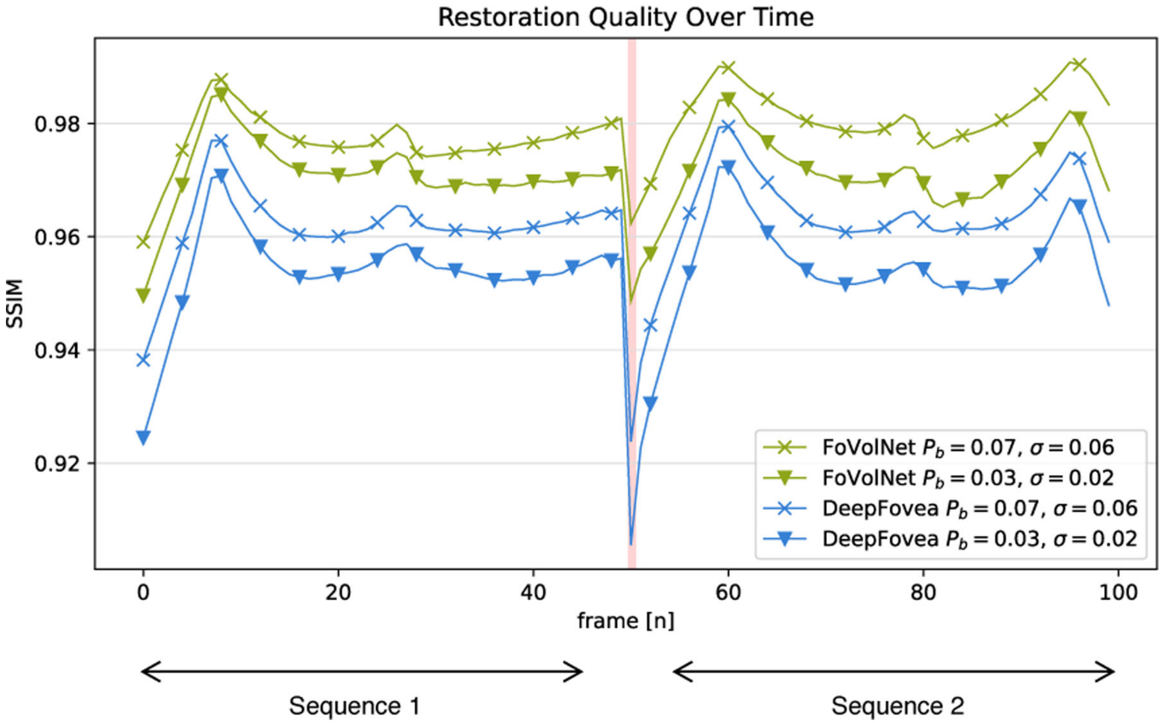

**Fig. 13:**

Structural similarity (SSIM) over the course of two 50-frame clips from the CHAMELEON dataset.

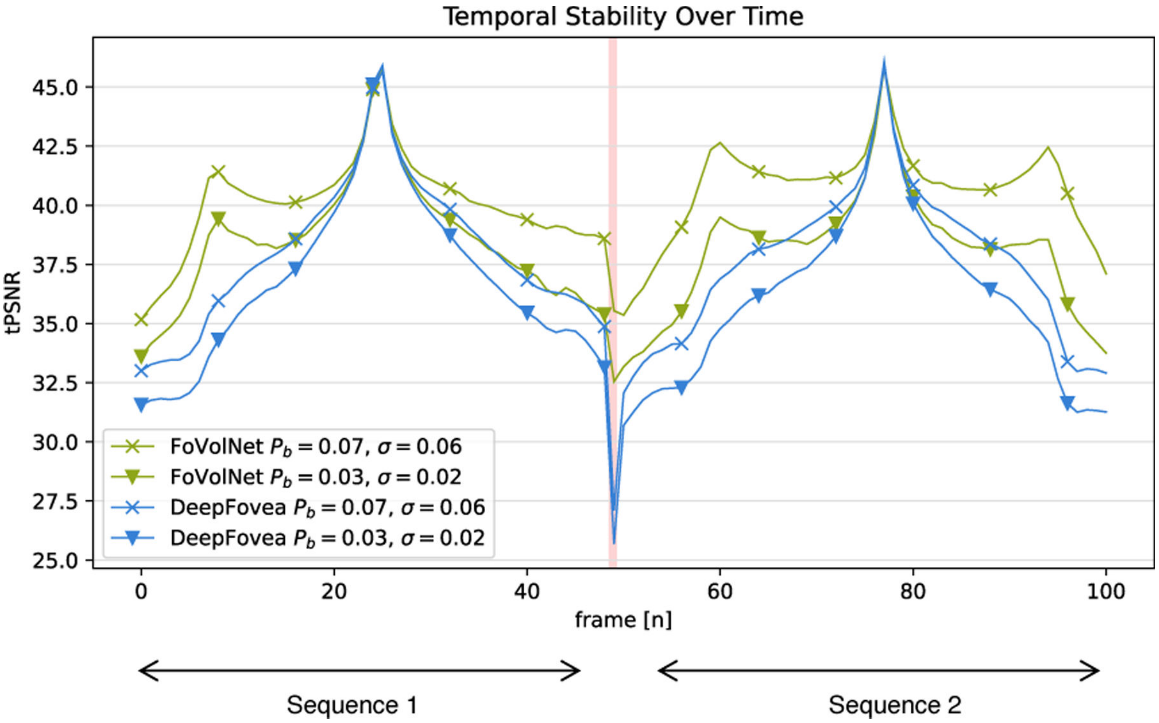

**Fig. 14:**

Temporal stability as measured by tPSNR [15] over the course of two 50-frame clips from the CHAMELEON dataset.

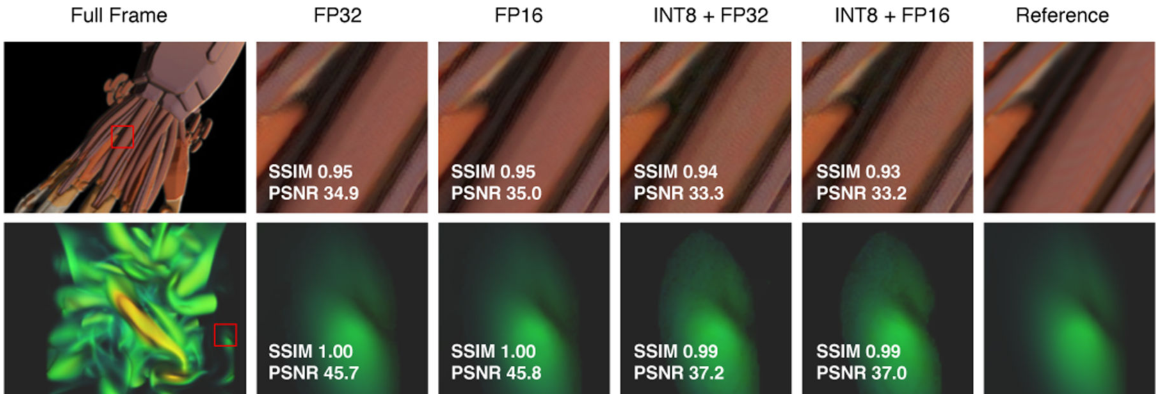

**Fig. 15:**

Comparison of reconstruction quality at different model precisions. Differences are most apparent on homogeneous surfaces and along subtle color gradients.

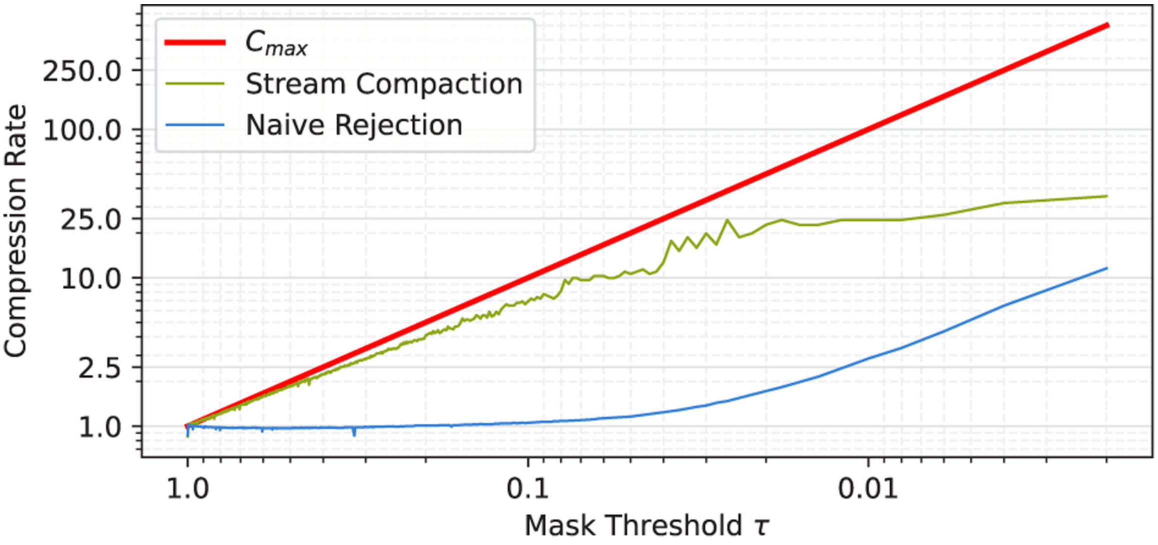

**Fig. 16:**

Compression efficiency of our stream compaction rendering technique. Data was recorded for the whole value spectrum of sampling threshold $\tau$. A curve for $C_{max}$ shows the maximum achievable efficiency at any given threshold value.

**Table 1:**

Datasets used to train and evaluate FoVolNet. Using our ray marching renderer, each volume was rendered as a camera fly-through sequence around the object. The skull and chameleon datasets were used for evaluation and were not part of the training datasets.

| Dataset | Dimensions | Data Type |
|---|---|---|
| SKULL [30] | 256*x*256*x*256 | uint8 |
| CHAMELEON [23, 34] | 1024*x*1024*x*1080 | uint8 |
| MECHANICAL HAND | 640*x*220*x*229 | float32 |
| VORTICES 1 [50] | 128*x*128*x*128 | float32 |
| VORTICES 2 [50] | 128*x*128*x*128 | float32 |
| SUPERNOVA [39] | 432*x*432*x*432 | float32 |

**Table 2:**

List of image augmentations applied during training. All augmentations affect the whole sequence of images to not introduce any unwanted combinations of effects.

| **Name** | **Description** | P(x) |
|---|---|---|
| Colors | Randomly permutes color channels | 0.6 |
| Flip Horizontal | Flips whole sequence along y axis | 0.5 |
| Flip Vertical | Flips whole sequence along x axis | 0.5 |
| Grayscale | Converts RGB input to grayscale | 0.3 |
| Static | Turns a real sequence into a number of static frames | 0.3 |
| Padding | Pads the whole sequence by a random number of pixels | 0.1 |

## Table 3:

Relative speed-up times when compared to the baseline raymarching renderer. The data is based on that shown in Figure 10. $P_b$ is given and $P_f$ was calculated using the listed $\sigma$ values.

| **Dataset** | **FoVolNet**<br>$P_b = 0.03$<br>$\sigma = 0.02$ | **FoVolNet**<br>$P_b = 0.07$<br>$\sigma = 0.06$ | **DeepFovea**<br>$P_b = 0.03$<br>$\sigma = 0.02$ | **DeepFovea**<br>$P_b = 0.07$<br>$\sigma = 0.06$ |
|---|---|---|---|---|
| VORT. 1 | 3.21× | 2.40× | 2.42× | 1.98× |
| VORT. 2 | 3.80× | 2.89× | 2.88× | 2.34× |
| SKULL | 3.12× | 2.19× | 2.45× | 1.88× |
| MECH. HAND | 2.66× | 1.74× | 2.23× | 1.55× |
| CHAM. | 3.22× | 2.45× | 3.05× | 1.73× |
| SUPERN. | 3.67× | 2.09× | 3.44× | 2.08× |
| **Overall** | **3.28×** | **2.29×** | **2.75×** | **1.93×** |

**Table 4:**

Just-objectionable-difference scores [35] of reconstruction output from the CHAMELEON dataset computed at different thresholds.

| $\tau$ | JOD Score FoVolNet | JOD Score DeepFovea |
|---|---|---|
| 0.10 | 9.35 | 8.53 |
| 0.07 | 9.28 | 8.47 |
| 0.03 | 8.86 | 8.21 |
| 0.01 | 8.26 | 7.51 |