Stable View Synthesis

Gernot Riegler Vladlen Koltun
Intel Labs

Figure 1: Stable View Synthesis synthesizes spatially and temporally coherent photorealistic views of complex real-world scenes. Top and left: new views of scenes from the Tanks and Temples dataset [19]. Bottom right: a new view of a scene from the FVS dataset [30].

Abstract

We present Stable View Synthesis (SVS). Given a set of source images depicting a scene from freely distributed viewpoints, SVS synthesizes new views of the scene. The method operates on a geometric scaffold computed via structure-from-motion and multi-view stereo. Each point on this 3D scaffold is associated with view rays and corresponding feature vectors that encode the appearance of this point in the input images. The core of SVS is view-dependent on-surface feature aggregation, in which directional feature vectors at each 3D point are processed to produce a new feature vector for a ray that maps this point into the new target view. The target view is then rendered by a convolutional network from a tensor of features synthesized in this way for all pixels. The method is composed of differentiable modules and is trained end-to-end. It supports spatially-varying view-dependent importance weighting and feature transformation of source images at each point; spatial and temporal stability due to the smooth dependence of on-surface feature aggregation on the target view; and synthesis of view-dependent effects such as specular reflection. Experimental results demonstrate that SVS outperforms state-of-the-art view synthesis methods both quantitatively and qualitatively on three diverse real-world datasets, achieving unprecedented levels of realism in free-viewpoint video of challenging large-scale scenes. Code is available at https://github.com/intel-isl/StableViewSynthesis

1. Introduction

Photorealistic view synthesis can allow us to explore magnificent sites in faraway lands without leaving the comfort of our homes. This requires advancing the technology towards two key goals. First, the synthesized images should be photorealistic: indistinguishable from reality. Second, the user should be free to move through the scene, as in the real world, exploring it from any physically realizable viewpoint.

In this paper, we present a new method for photorealistic view synthesis that brings these two goals closer. Our input is a set of images that can be taken for example from a handheld video of the scene. From these images, we construct a 3D geometric scaffold via off-the-shelf structure-from-motion, multi-view stereo, and meshing. Input images are encoded by a convolutional network and the resulting deep features are mapped onto the geometric scaffold. As a result, for any point on the scaffold, we can obtain a collection of view rays with associated feature vectors, which correspond to input images that see this point.

The core of our method is an approach to synthesizing arbitrary new views given this representation of the scene. Each pixel in the new view is mapped onto the geometric scaffold to obtain the set of input rays with associated feature vectors, and an output ray towards the new view. The feature vectors from the input rays are then aggregated, taking the geometry of the input and output rays into account, by a differentiable module that produces a feature vector for the output ray. Together, the feature vectors synthesized for all pixels form a feature tensor. The new image is rendered from this feature tensor by a convolutional network.

All steps of the method are differentiable and the complete pipeline can be trained end-to-end to maximize photorealism. All steps can be implemented efficiently, leveraging parallelism across pixels. Crucially, the computation of a feature vector for a new output ray does not require any heuristic selection of input rays. The computation aggregates information from all input rays in a differentiable module that is informed by the spatial layout of the rays and is optimized end-to-end. This supports temporal stability for smoothly moving viewpoints.

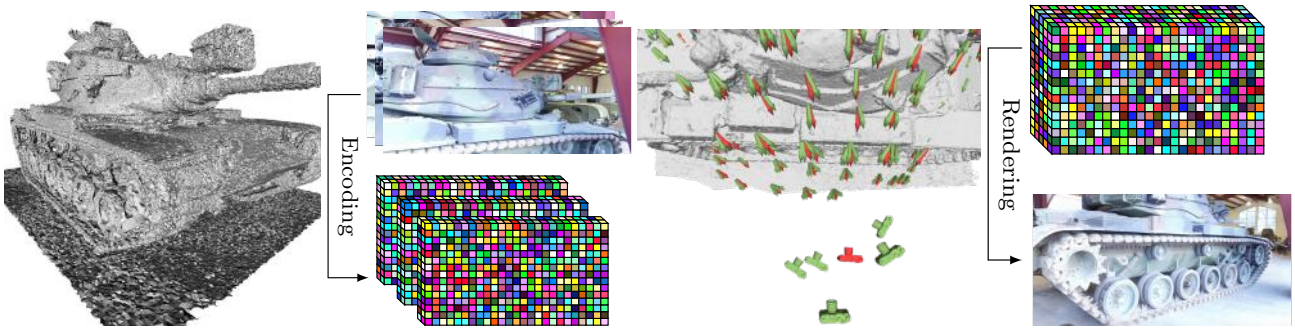
We evaluate the presented method on three diverse datasets of real scenes and objects: Tanks and Temples [19], FVS [30], and DTU [1]. Tanks and Temples and FVS provide handheld video sequences of large real-world scenes; the objective is to use these video sequences as input to enable photorealistic rendering of the scenes from new views. DTU provides regularly-spaced outside-in images of challenging real objects. On all three datasets, SVS convincingly outperforms the state of the art. On Tanks and Temples, our method reduces the LPIPS error for new views by up to 10 absolute percentage points (a reduction of roughly 30% on average) relative to the prior state of the art, while also improving PSNR and SSIM. On the FVS dataset, our method likewise outperforms the state of the art on all metrics, reducing LPIPS by 7 absolute percentage points on average relative to the best prior method. On DTU, we set the new state of the art for novel view synthesis, attaining an average LPIPS error of 4.5% over the test scenes in extrapolation mode and 1.6% for view interpolation. A number of our synthesized images for new views in Tanks and Temples and FVS scenes are shown in Figure 1, and video sequences are provided in the supplementary video.

2. Related Work

Image-based rendering has a long history in computer vision and graphics. Shum and Kang [34] provide a review of early approaches and foundational work. More recent highlights include the work of Wood et al. [43], Buehler et al. [4], Davis et al. [10], Chaurasia et al. [5], Kopf et al. [20], Hedman et al. [16], and Penner and Zhang [28].

More recently, deep learning techniques have enabled a new level of flexibility and realism. Given a geometric reconstruction of the scene, Hedman et al. [15] map image mosaics to the target view and refine them via a blending network. Thies et al. [40] learn image-dependent effects via a convolutional network. Choi et al. [7] warp volumetric information from the source images to the target view. Riegler and Koltun [30] warp features from a heuristically selected set of source images into the target view and blend them using a recurrent convolutional network. Other approaches directly learn features for each 3D point [2, 9] or vertex [39] of a geometric reconstruction.

Our method is most closely related to the Free View Synthesis approach of Riegler and Koltun [30], in that both methods operate on a geometric scaffold obtained via SfM, MVS, and meshing, and both methods utilize encoder and decoder networks to encode input images into feature tensors and render the new view from a new feature tensor, respectively. However, the methods differ crucially at their core: the synthesis of the feature tensor for the new view. The FVS pipeline heuristically selects a set of relevant source images for a given target view, warps the feature tensors from these input views into the target camera frame, and blends these warped feature tensors via a recurrent convolutional network. The heuristic selection of relevant input views leads to temporal instability when the set of selected views changes and causes drastic visual artifacts when the selected views do not contain all the information needed to cover some part of the output image. Furthermore, the sequential ordering of the input feature tensors processed by the recurrent network is artificial and can lead to instability when it changes. In contrast, SVS synthesizes feature vectors for the new view on the 3D surface itself, taking all input images into account as needed, and using set operators rather than sequence models to avoid arbitrary or-



(a) Geometric scaffold (b) Encoding source images (c) On-surface aggregation (d) Rendering in the target view

Figure 2: **Overview of Stable View Synthesis.** (a) A geometric scaffold of the scene is constructed using structure-from-motion, multiple-view stereo, and meshing. (b) All source images are encoded into feature tensors via a convolutional network. (c) Given a new target view (red camera), feature vectors from the source images (green cameras) are aggregated on the geometric scaffold. Red arrows map 3D points to the target view, green arrows map the same points to the source views. (d) The output image in the target view is rendered from a tensor of synthesized feature vectors by a convolutional network.

dering. There is no heuristic selection of relevant images, no temporal instability due to changes in this set, no drastic artifacts due to the heuristic omission of relevant information, and no instability due to shifts in sequential processing. All processing takes all available information into account as needed, via permutation-invariant set operators, in a pipeline that is composed entirely of differentiable modules that are trainable end-to-end.

Several methods incorporate concepts similar to plane-sweep volumes [8] into the network architecture to synthesize novel views. Flynn et al. [13] utilize this concept to interpolate between views. Kalantari et al. [17] use this idea for a light-field setup with a fixed number of cameras. Additional directional lighting extensions to these architectures enable synthesis of complex appearance effects [3, 44].

Multi-plane images (MPIs) [48] are also often used in conjunction with deep networks [47]. Here the image is represented by color+α planes at different depths and novel views can be rendered back-to-front. Srinivasan et al. [38] show that a limiting factor in MPIs is the depth resolution and propose a randomized-resolution training procedure. This work is extended by Mildenhall et al. [24] who use multiple local MPIs and practical user guidance. Flynn et al. [12] train a network to predict high-quality MPIs via learned gradient descent. Li et al. [21] extend this line of work to image sets with strong appearance variation.

Another class of methods utilizes volumetric representations. Sitzmann et al. [36] lift 2D image features to a common 3D volume. The features are synthesized via a scene-dependent rendering network. To overcome the memory requirements of voxel-based representations, Lombardi et al. [22] learn a dynamic irregular grid structure. In Scene Representation Networks [37], the volume is represented as an MLP and images are rendered via differentiable ray marching. Niemeyer et al. [26] build upon an implicit occupancy representation that can be trained by posed images via implicit differentiation. Neural Radiance Fields [25] produce impressive results by training an MLP that maps 3D rays to occupancy and color. Images are synthesized from this representation via volume rendering. This methodology has been extended to unbounded outdoor scenes [45] and crowdsourced image collections [23].

3. Overview

A visual overview of SVS is provided in Figure 2. Our input is a set of source images $\{\mathcal{I}_n\}_{n=1}^N$, which are used to erect a geometric scaffold Γ and are the basis for the on-surface feature representation. Given a new viewpoint $(\mathbf{R}_t, \mathbf{t}_t)$ and camera intrinsics $\mathbf{K}_t$, our goal is to synthesize an image $\mathcal{O}$ that depicts the scene in this new view.

Preprocessing: Our method leverages a 3D geometric scaffold. To construct this scaffold, we use standard structure-from-motion, multi-view stereo, and surface reconstruction [32, 33]. We first run structure-from-motion [32] to get camera intrinsics $\{\mathbf{K}_n\}_{n=1}^N$ and camera poses as rotation matrices $\{\mathbf{R}_n\}_{n=1}^N$ and translation vectors $\{\mathbf{t}_n\}_{n=1}^N$. In the rest of the paper, we use $\{\mathcal{I}_n\}_{n=1}^N$ to denote the rectified images after structure-from-motion. We then run multi-view stereo on the posed images, obtain per-image depthmaps, and fuse these into a point cloud. Delaunay-based 3D surface reconstruction is applied to this point cloud to get a 3D surface mesh Γ. We use COLMAP [32, 33] for preprocessing in all experiments, but our method can utilize other SfM and MVS pipelines.

In addition, each image $\mathcal{I}_n$ is encoded by a convolutional network to obtain a feature tensor $\mathcal{F}_n$, which provides a feature vector for each pixel in $\mathcal{I}_n$.

View synthesis: To synthesize the new view $\mathcal{O}$, we back-

project pixels in $\mathcal{O}$ onto the scaffold Γ. For each point $\mathbf{x} \in \Gamma$ obtained in this way, we query the set of input images in which $\mathbf{x}$ is visible. For each such image $\mathcal{I}_k$, we obtain a feature vector $\mathbf{f}_k$ along the corresponding ray $\mathbf{v}_k$ to $\mathbf{x}$. See Figure 3 for an illustration. The set $\{(\mathbf{v}_k, \mathbf{f}_k)\}_k$ of view rays with corresponding feature vectors is then processed by a differentiable set network that is conditioned on the output view direction $\mathbf{u}$. This network produces a new feature vector $\mathbf{g}$. Feature vectors $\mathbf{g}$ are obtained in this way for all pixels in $\mathcal{O}$. The resulting feature tensor $\mathcal{G}$ is decoded by a convolutional network to produce the output image.

Note that SVS differs from works that use neural point features [2, 9] or neural mesh textures [39], which fit feature vectors from scratch (initialized with random noise) per scene on a point cloud or mesh. SVS also differs from methods that project full (encoded) source images to the target view [15, 30]; in SVS, each 3D point independently aggregates features from a different set of source images.

4. Feature Processing and Aggregation

Image encoding: Each source image $\mathcal{I}_n$ is encoded into a feature tensor by a convolutional network based on the U-Net architecture [31]. This network is denoted by ϕ_{enc}. The encoder part of ϕ_{enc} consists of an ImageNet-pretrained ResNet18 [14], where we freeze the BatchNorm parameters. In the decoder part of ϕ_{enc}, each stage upsamples the feature map using nearest-neighbor interpolation, concatenates it with the corresponding feature map (of the same resolution) from the encoder, and applies convolution and activation layers. We denote the feature tensor produced by this network by $\mathcal{F}_n = \phi_{\text{enc}}(\mathcal{I}_n)$.

On-surface aggregation: The core of our method is the computation of a target feature vector $\mathbf{g}(\mathbf{x}, \mathbf{u})$ for each point $\mathbf{x} \in \Gamma \subset \mathbb{R}^3$ on the 3D geometric scaffold. This feature vector is computed as a function of the viewing direction $\mathbf{u}$ from the target camera center to the surface point $\mathbf{x}$, and tuples $\{(\mathbf{v}_k, \mathbf{f}_k(\mathbf{x}))\}_{k=1}^K$. Here $\{\mathbf{f}_k(\mathbf{x})\}_{k=1}^K$ are source image features that correspond to $\mathbf{x}$ in the image encodings $\{\mathcal{F}_k\}_{k=1}^K$ in which $\mathbf{x}$ is visible, and $\{\mathbf{v}_k\}_{k=1}^K$ are the corresponding viewing directions. Specifically, $\mathbf{f}_k(\mathbf{x}) = \mathcal{F}_k(\mathbf{K}_k(\mathbf{R}_k\mathbf{x} + \mathbf{t}_k))$ using bilinear interpolation.

More formally, the target feature vector for a given 3D surface point $\mathbf{x}$ is computed as

$$\mathbf{g}(\mathbf{x}, \mathbf{u}) = \phi_{\text{aggr}}(\mathbf{u}, \{(\mathbf{v}_k, \mathbf{f}_k(\mathbf{x}))\}_{k=1}^K)\,, \tag{1}$$

where K is the number of source images that $\mathbf{x}$ is visible in and ϕ_{aggr} is an aggregation function. The function ϕ_{aggr} must fulfill a number of criteria; most notably, it should be differentiable and must process any number K of input features, in any order. We explore multiple designs based on differentiable set operators and select one of them based on empirical performance (reported in Section 6).

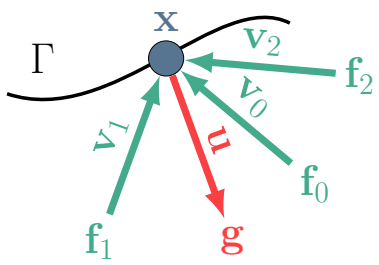


Figure 3: **On-surface aggregation.** A 3D point $\mathbf{x}$ on the geometric scaffold Γ is seen in a set of source images. Each such image contributes a feature vector $\mathbf{f}_k$ along a ray $\mathbf{v}_k$ (green). On-surface aggregation uses a differentiable set network to process this data and produces a feature vector $\mathbf{g}$ for the target ray $\mathbf{u}$ (red).

A simple choice for ϕ_{aggr} is a weighted average, where the weights are based on the alignment between the source and target directions:

$$\phi_{\text{aggr}}^{\text{WA}} = \frac{1}{W}\sum_{k=1}^{K} \max(0, \mathbf{u}^T\mathbf{v}_k)\mathbf{f}_k(\mathbf{x})\,. \tag{2}$$

Here $W = \sum_{k=1}^K \max(0, \mathbf{u}^T\mathbf{v}_k)$ is the sum of all weights. For a more expressive aggregation function, we can leverage PointNet [29]. Specifically, we concatenate the source and target directions to the source features, apply an MLP to each feature vector, and aggregate the results:

$$\phi_{\text{aggr}}^{\text{MLP}} = \nu_{k=1}^K \,\text{MLP}(\mathbf{f}'_k)\,. \tag{3}$$

Here $\mathbf{f}'_k = [\mathbf{u}, \mathbf{v}_k, \mathbf{f}_k(\mathbf{x})]$ is the concatenation of source and target directions with the feature vector, and ν is a permutation-invariant operator such as mean or $\max$. Instead of an MLP, we can also use a graph attention network (GAT) [41] that operates on a fully-connected graph between the source views per 3D point:

$$\phi_{\text{aggr}}^{\text{GAT}} = \nu_{k=1}^K \,\text{GAT}\left(\{\mathbf{f}'_k\}_{k=1}^K\right)\big|_k\,, \tag{4}$$

where $\cdot|_k$ is the readout of the feature vector on node k.

Aggregation functions presented so far compute the target feature $\mathbf{g}$ as a set feature. Another possibility is to read out the target feature vector at the target viewing direction $\mathbf{u}$. Specifically, we can create a fully connected graph over source features $\{[\mathbf{v}_k, \mathbf{f}_k]\}_{k=1}^K$ and an initial target feature $[\mathbf{u}, \mathbf{g}']$, where $\mathbf{g}'$ is initialized via Equation (2). Then we can define the readout aggregation function as

$$\phi_{\text{aggr}}^{\text{GAT-RO}} = \text{GAT}\left(\{[\mathbf{u}, \mathbf{g}']\} \cup \{[\mathbf{v}_k, \mathbf{f}_k(\mathbf{x})]\}_{k=1}^K\right)\big|_0\,, \tag{5}$$

where $\cdot|_0$ denotes the readout of the feature vector associated with the target node.

Rendering: We now describe how the surface points $\mathbf{x}$ are obtained and how the output image $\mathcal{O}$ in the target view is rendered. Given a user-specified camera $\mathbf{K}_t$

and new camera pose $(\mathbf{R}_t, \mathbf{t}_t)$, we compute a depth map $\mathcal{D} \in \mathbb{R}^{H \times W}$ from the proxy geometry Γ. We then unproject each pixel center of the target view back to 3D based on the depth map $\mathcal{D}$, obtaining a surface point for each pixel in $\mathcal{O}$, $\{\mathbf{x}_{h,w}\}_{h,w=1,1}^{H \times W}$. Note that $\mathcal{D}$ may not have valid depth values for some pixels due to incompleteness of the surface mesh Γ, or for background regions such as the sky. We use ∞ as the depth value for such pixels.

Given the 3D surface points $\{\mathbf{x}_{h,w}\}_{h,w=1,1}^{H \times W}$, we can compute view-dependent feature vectors $\{\mathbf{g}(\mathbf{x}_{h,w})\}_{h,w=1,1}^{H \times W}$ as described above and assemble a feature tensor $\mathcal{G} = [\mathbf{g}_{h,w}]_{h,w=1,1}^{H \times W}$. For 3D surface points $\mathbf{x}_{h,w}$ that do not map to any source image, we set $\mathbf{g}_{h,w}$ to $\mathbf{0}$.

To synthesize the image $\mathcal{O}$ from the feature tensor $\mathcal{G}$, we use a convolutional network, denoted by ϕ_{render}: $\mathcal{O} = \phi_{\text{render}}(\mathcal{G})$. The main goal of this network is to regularize the feature map, for example to counteract scale and exposure differences in the source images, and to inpaint missing regions. For this purpose, we use a sequence of L U-Nets, where each U-Net learns the residual to its input: $\phi_{\text{render}}(\mathcal{G}) = \phi_{\text{render}}^{L}(\mathcal{G} + \phi_{\text{render}}^{L-1}(\mathcal{G} + \dots))$.

5. Training

Training a scene-agnostic model: We train the three networks (ϕ_{enc}, ϕ_{aggr}, and ϕ_{render}) end-to-end. Given a set of scenes, we first sample a scene and a source image $\mathcal{I}_n$ that will serve as ground truth. From the remaining source images of the sampled scene, we sample a subset of M source images used for one training pass. We then minimize a perceptual loss that is inspired by Chen and Koltun [6]:

$$\mathbb{L}(\mathcal{O}, \mathcal{I}_n) = ||\mathcal{O} - \mathcal{I}_n||_1 + \sum_l \lambda_l ||\phi_l(\mathcal{O}) - \phi_l(\mathcal{I}_n)||_1 \,, \quad (6)$$

where ϕ_l are the outputs of the layers 'conv1_2', 'conv2_2', 'conv3_2', 'conv4_2', and 'conv5_2' of a pretrained VGG-19 network [35]. We use Adam [18] with a learning rate of 10^{-4} and set $\beta_1 = 0.9$, $\beta_2 = 0.9999$, and $\epsilon = 10^{-8}$ to train the network.

Network fine-tuning: The scene-agnostic training procedure described above yields a general network that can be applied to new scenes without retraining or fine-tuning. However, scenes we apply our method to can be very different from scenes we train on: for example, training the network on Tanks and Temples and applying it on DTU. We could follow common practice and fine-tune the network parameters $\theta = [\theta_{\text{enc}}, \theta_{\text{aggr}}, \theta_{\text{render}}]$ on source images of the target scene, which are provided as input. Starting from the trained scene-agnostic model, we apply the same training procedure as described above, but only sample training images $\mathcal{I}_n$ from the source images of the target scene.

Scene fine-tuning: An even more powerful form of fine-tuning is to optimize not only the network parameters but also parameters associated with the source images. This enables the optimization to harmonize inconsistencies across images, such as different exposure intervals due to auto-exposure, image-specific motion blur, and other aberrations in the source images.

Recall that so far we have optimized the objective $\min_\theta \mathbb{L}(\mathcal{O}, \mathcal{I}_n)$, where $\theta = [\theta_{\text{enc}}, \theta_{\text{aggr}}, \theta_{\text{render}}]$ are the parameters of the encoder, aggregation, and rendering networks. Note also that the output image $\mathcal{O}$ produced by the networks is a function of the encoded source images $\{\phi_{\text{enc}}(\mathcal{I}_m; \theta_{\text{enc}})\}_{m=1}^{M}$.

So far, the image encoder ϕ_{enc} took the source image $\mathcal{I}_m$ as input, but the training process only optimized the network parameters θ_{enc}. The key idea of our more powerful fine-tuning is to also optimize the source images $\{\phi_{\text{enc}}(\mathcal{I}_m; \theta_{\text{enc}})\}_{m=1}^{M}$ that are used as input. (Importantly, the optimization cannot alter the image $\mathcal{I}_n$ that is used as ground truth in the loss $\mathbb{L}(\mathcal{O}, \mathcal{I}_n)$.) Specifically, we change the image encoder to $\phi_{\text{enc}}(m; \theta_{\text{enc}}, \theta_{\text{imgs}})$, i.e., the input of the network changes from a source image $\mathcal{I}_m$ to the index m, which is used by the network to index into a pool of trainable parameters θ_{imgs} that are initialized with the actual source images. The source images have become mutable and can be optimized during the training process. The encoder can also be denoted by $\phi_{\text{enc}}(\theta_{\text{imgs}}[m]; \theta_{\text{enc}})$ to establish the connection to the original encoder.

The optimization objective becomes $\min_{\theta, \theta_{\text{imgs}}} \mathbb{L}(\mathcal{O}, \mathcal{I}_n)$. Aside from the modified objective, the training procedure stays the same. Note that θ_{imgs} are initialized with the source images $\{\mathcal{I}_n\}_{n=1}^{N}$, but the original, unmodified source images $\{\mathcal{I}_n\}_{n=1}^{N}$ are used throughout the training process in the loss $\mathbb{L}(\mathcal{O}, \mathcal{I}_n)$. Thus the optimization process is forced to produce output $\mathcal{O}$ that matches the *original* images $\mathcal{I}_n$ and cannot degenerate to a trivial solution such as setting all the source images to a uniform color. The optimization over θ_{imgs} merely gives the training process the flexibility to modify its perceived input images (e.g., regularizing away inconsistencies) to be able to more closely match the immutable ground-truth targets.

6. Evaluation

We begin by evaluating our architectural choices in a set of controlled experiments. We then compare SVS to the state of the art on three challenging datasets: Tanks and Temples [19], the FVS dataset [30], and DTU [1]. We use the same Tanks and Temples scenes for training as Riegler and Koltun [30] with the difference that *Ignatius* and *Horse* are withheld for validation, to get a clean split between training, validation, and test scenes. Thus 15 of the 21 Tanks and Temples scenes are used for training, 2 for validation, and 4 for evaluation. We implement the networks in PyTorch [27] and train the scene-agnostic model for

	↑PSNR	↑SSIM	↓LPIPS%
Weighted Mean	**21.42**	**0.870**	12.84
MLP Mean	**21.25**	**0.869**	**12.51**
MLP Max	20.95	**0.863**	12.65
GAT Mean	21.01	**0.864**	12.84
GAT Max	21.05	**0.864**	13.09
GAT Readout	20.88	**0.862**	12.81

(a) 3D aggregation function

	↑PSNR	↑SSIM	↓LPIPS%
1	21.20	**0.868**	12.62
3	21.25	**0.869**	12.51
5	21.30	**0.870**	12.46
7	**21.55**	**0.872**	12.41
9	**21.39**	**0.871**	**12.27**

(b) Number of refinement steps

	↑PSNR	↑SSIM	↓LPIPS%
RGB Averaging	21.15	0.844	22.84
Network FT w/o PT	21.13	0.865	15.05
General	21.59	**0.872**	12.19
Network FT	**22.16**	**0.874**	11.26
Scene FT	**22.02**	**0.873**	**9.99**

(c) Fine-tuning

Table 1: **Controlled experiments.** Mean accuracy over the validation scenes. Numbers in bold are within 1% of the best.

General Network FT Scene FT

Figure 4: **The impact of fine-tuning.** The figure shows a new target view that was not seen by the network during fine-tuning.

600,000 iterations with a batch size of 1, sampling $M = 3$ source images per iteration. We use three image fidelity metrics: LPIPS [46] (reported in percent), which has been shown to correlate well with human perception, alongside SSIM [42] and PSNR, which are metrics that are more attuned to low-level image differences.

Architectural choices: In the first set of controlled experiments, we validate our architectural choices. As outlined above, we train on 15 Tanks and Temples scenes and validate on the 2 withheld scenes.

First, we compare a set of different 3D aggregation functions. The results are summarized in Table 1a. The first row reports the accuracy with the *Weighted Mean* aggregation as described in Equation (2). The second and third rows report accuracy with the *MLP* aggregation function (see Equation (3)), once with the mean and once with the max pooling operator. Rows four and five report accuracy with the graph attention network aggregation as described in Equation (4), again once with mean and once with max pooling of the GAT feature vectors. The last row reports accuracy with the $\phi_{\text{aggr}}^{\text{GAT-RO}}$ aggregation function as defined in Equation (5). The results give a slight edge to *MLP Mean* aggregation, in particular for the LPIPS metric, which correlates most reliably with human perception. We therefore adopt this aggregation function for the other experiments.

In the second experiment, we want to verify that the rendering network benefits from multiple refinement stages. We thus vary the number L of residual U-Net stages in ϕ_{render}. The results are reported in Table 1b. We observe that there is no significant difference in terms of PSNR and SSIM, but LPIPS decreases with the number of refinement stages. We thus set $L = 9$ for the other experiments.

In the third controlled experiment, we evaluate the impact of scene-specific fine-tuning. Table 1c summarizes the results. In the first row we show a simple baseline that just averages the RGB values per 3D point and in the second row the network is only trained on the source images of the test scene (not trained on the pre-training scenes). The third row reports the accuracy of the scene-agnostic network, which is trained on the 15 training scenes from Tanks and Temples and is not fine-tuned on the validation scenes. The fourth row reports the accuracy of the same network after fine-tuning the network weights on the source images of the target scene. (Only the source images are used for fine-tuning. Target views that are used for evaluation are never used during training or fine-tuning.) The fifth row reports the accuracy of the network after fine-tuning both the network weights and the input images, as described in Section 5. Although none of the fine-tuning methods significantly alters PSNR or SSIM, we can see a clear improvement in LPIPS. We thus use scene fine-tuning for all other experiments. Figure 4 shows the effect of fine-tuning on an example image.

Tanks and Temples dataset: We now compare SVS to the state of the art on four new scenes (not used for training or validation) from the Tanks and Temples dataset [19], following the protocol of Riegler and Koltun [30]. For each scene, there is a specific set of source images and a disjoint set of target views for evaluation.

We compare to a variety of recent methods that represent different approaches to view synthesis and have been applied in comparable settings in the past. For Local Light Field Fusion (*LLFF*) [24] we used the publicly available code. Since no training code is available, we use the provided pretrained network weights. For Extreme View Synthesis (*EVS*) [7] we also use the publicly available code and

	Truck			M60			Playground			Train		
	↑PSNR	↑SSIM	↓LPIPS%	↑PSNR	↑SSIM	↓LPIPS%	↑PSNR	↑SSIM	↓LPIPS%	↑PSNR	↑SSIM	↓LPIPS%
LLFF [24]	10.78	0.454	60.62	8.98	0.431	71.76	14.40	0.578	53.93	9.15	0.384	67.40
EVS [7]	14.22	0.527	43.52	7.41	0.354	75.71	14.72	0.568	46.85	10.54	0.378	67.62
NPBG [2]	21.88	0.877	15.04	12.35	0.716	35.57	23.03	**0.876**	16.65	18.08	0.801	25.48
NeRF [25]	20.85	0.738	50.74	16.86	0.701	60.89	21.55	0.759	52.19	16.64	0.627	64.64
NeRF++ [45]	22.77	0.814	30.04	18.49	0.747	43.06	22.93	0.806	38.70	17.77	0.681	47.75
FVS [30]	22.93	0.873	13.06	16.83	0.783	30.70	22.28	0.846	19.47	18.09	0.773	24.74
Ours w/o FT	23.09	**0.893**	12.41	19.41	**0.827**	23.70	**23.61**	**0.876**	17.38	18.42	0.809	19.42
Ours	**23.86**	**0.895**	**9.34**	**19.97**	**0.833**	**20.45**	**23.72**	**0.884**	**14.22**	**18.69**	**0.820**	**15.73**

Table 2: **Accuracy on Tanks and Temples.** Accuracy on the test scenes. Numbers in bold are within 1% of the best.

Figure 5: **Qualitative results on Tanks and Temples.** Comparison of SVS to the best-performing prior methods.

the provided network weights. Neural Point Based Graphics (*NPBG*) [2] is fitted per scene using the published code and pretrained rendering network weights. For Neural Radiance Fields (*NeRF*) [25] and *NeRF++* [45] we manually define the bounding volume around the main object in each scene. These approaches are trained per scene. For Free View Synthesis (*FVS*) [30] we use the publicly available code and the published network weights, which had been trained on the union of our training and validation scenes.

The results are summarized in Table 2. As observed in prior work [30], *LLFF* and *EVS* struggle in this challenging view synthesis setting. We also see that *NeRF++* improves over *NeRF*, but neither attain the accuracy of the best-performing methods. SVS without any scene-specific fine-tuning (*Ours w/o FT*) already outperforms all prior work for most scenes, especially with respect to LPIPS. Our full method (*Ours*) achieves the best results on all scenes.

Figure 5 shows images synthesized by the best-performing methods on a number of scenes. *FVS* sometimes fails to utilize all the relevant images, which leads to miss-

	Bike		Flowers		Pirate		Digger		Sandbox		Soccertable	
	↑SSIM	↓LPIPS%	↑SSIM	↓LPIPS%	↑SSIM	↓LPIPS%	↑SSIM	↓LPIPS%	↑SSIM	↓LPIPS%	↑SSIM	↓LPIPS%
NPBG [2]	0.616	31.08	0.553	48.47	0.592	45.71	0.686	29.08	0.650	35.91	0.723	29.97
NeRF++ [45]	0.715	27.01	0.816	30.30	0.712	41.56	0.657	34.69	0.842	23.08	0.889	20.61
FVS [30]	0.592	27.83	0.778	26.07	0.685	35.89	0.668	23.27	0.770	30.20	0.819	19.41
Ours w/o FT	0.745	21.18	**0.848**	21.41	0.752	**29.21**	0.782	18.00	0.850	21.48	0.895	14.79
Ours	**0.757**	**20.84**	**0.845**	**20.82**	**0.760**	30.83	**0.791**	**16.12**	**0.862**	**20.00**	**0.912**	**13.07**

Table 3: **Accuracy on the FVS dataset.** Numbers in bold are within 1% of the best.

	65			106			118		
	↑PSNR	↑SSIM	↓LPIPS%	↑PSNR	↑SSIM	↓LPIPS%	↑PSNR	↑SSIM	↓LPIPS%
LLFF [24]	22.48/22.07	0.935/0.921	9.38/12.71	24.10/24.63	0.900/0.886	13.26/13.57	28.99/27.42	0.928/0.922	9.69/10.99
EVS [7]	23.26/14.43	0.942/0.848	7.94/22.11	20.21/11.15	0.902/0.743	14.91/29.57	23.35/12.06	0.928/0.793	10.84/25.01
NPBG [2]	16.74/15.44	0.889/0.873	14.30/19.45	19.62/20.26	0.847/0.842	18.90/21.13	23.81/24.14	0.867/0.879	15.22/16.88
NeRF [25]	**32.00/28.12**	**0.984/0.963**	3.04/8.54	**34.45/30.66**	**0.975/0.957**	7.02/10.14	**37.36/31.66**	**0.985/0.967**	4.18/6.92
FVS [30]	30.44/25.32	**0.984/0.961**	2.56/7.17	32.96/27.56	**0.979**/0.950	2.96/6.57	35.64/29.54	**0.985/0.963**	1.95/6.31
Ours w/o FT	30.08/23.98	**0.983/0.960**	2.36/7.16	32.06/29.01	**0.978/0.959**	3.54/5.36	35.65/30.42	**0.986/0.966**	2.15/5.15
Ours	**32.13**/26.82	**0.986/0.964**	**1.70/5.61**	**34.30/30.64**	**0.983/0.965**	**1.93/3.69**	**37.27/31.44**	**0.988/0.967**	**1.30/4.26**

Table 4: **Accuracy on DTU.** Numbers in bold are within 1% of the best. In each column, numbers on the left are for view interpolation, right for extrapolation.

ing regions. *NeRF++* suffers from blurring and patterning in the output, although it sometimes reconstructs details that are missing in our geometric scaffold. While the results of *NPBG* can be very good, it sometimes introduces noticeable artifacts in parts of the scene. Images synthesized by SVS are overall sharper, more complete, more accurate, and more temporally stable than the prior work. Please see the supplementary video for sequences.

Free View Synthesis dataset: Next, we compare SVS with prior work on the FVS dataset [30]. This dataset contains 6 scenes, each of which was recorded at least twice. The first recording provides the source images and the other recordings serve as ground truth for novel target views. Quantitative results are summarized in Table 3 and qualitative results are provided in the supplement. Due to space constraints, we omit PSNR values here. SVS improves over prior work on all scenes, according to all metrics. Note that SVS reduces the LPIPS relative to the best prior method by at least 5 absolute percentage points in every scene.

DTU: Lastly, we compare SVS to prior approaches on the DTU dataset [1]. DTU scenes are captured with a regular camera layout, where 49 images are taken from an octant of a sphere. We follow the protocol of Riegler and Koltun [30], use the same scenes, and use the 6 central cameras to evaluate view interpolation and the 4 corner cameras to evaluate view extrapolation.

Quantitative results are summarized in Table 4 and qualitative results are provided in the supplement. *LLFF* and *EVS* achieve reasonable results on this dataset, indicating that this setup conforms much better to their modeling assumptions. *NPBG* struggles on this dataset, possibly due to the small number of images per scene (i.e., 39). *NeRF* excels on this dataset; we manually specified a tight bounding box around the object to maximize the accuracy of *NeRF*. The results of *FVS* are on par with *NeRF* with respect to SSIM and LPIPS. For our method, the scene-agnostic model, which was trained on Tanks and Temples and has never seen DTU-like scenes, is already surprisingly competitive, and the full SVS method sets the new state of the art for novel view synthesis on this dataset with respect to LPIPS, attaining an average LPIPS error of 4.5% in extrapolation mode and 1.6% for view interpolation.

7. Discussion

We presented a view synthesis method that is based on differentiable on-surface feature processing. The method aggregates deep features from source images adaptively on a geometric scaffold of the scene using a differentiable set network. The pipeline is trained end-to-end and learns to aggregate features from all images, obviating the need for heuristic selection of "relevant" source images. Our method sets a new state of the art for photorealistic view synthesis on large-scale real-world scenes.

There are a number of exciting avenues for future work. First, we look forward to continued progress in 3D reconstruction [19], which can further advance the fidelity of the images synthesized by the presented approach. Second, it would be interesting to extend the approach to image sets with strong appearance variation, perhaps enabling relighting of the scenes at test time [21, 23]. Lastly, the presented approach, like most recent view synthesis work, only handles static scenes. This enables the user to look at these environments but not engage and interact with them. An exciting challenge for the field is to enable interactive manipulation of such scenes while maintaining photorealism.

References

[1] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-Scale Data for Multiple-View Stereopsis. *IJCV*, 120(2), 2016. 2, 5, 8, 11

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural Point-Based Graphics. In *ECCV*, 2020. 2, 4, 7, 8, 12, 13

[3] Sai Bi, Zexiang Xu, Kalyan Sunkavalli, David Kriegman, and Ravi Ramamoorthi. Deep 3D Capture: Geometry and Reflectance from Sparse Multi-View Images. In *CVPR*, 2020. 3

[4] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured Lumigraph Rendering. In *SIGGRAPH*, 2001. 2

[5] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth Synthesis and Local Warps for Plausible Image-based Navigation. *SIGGRAPH*, 32, 2013. 2

[6] Qifeng Chen and Vladlen Koltun. Photographic Image Synthesis with Cascaded Refinement Networks. In *ICCV*, 2017. 5

[7] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme View Synthesis. In *ICCV*, 2019. 2, 6, 7, 8

[8] Robert T Collins. A Space-Sweep Approach to True Multi-Image Matching. In *CVPR*, 1996. 3

[9] Peng Dai, Yinda Zhang, Zhuwen Li, Shuaicheng Liu, and Bing Zeng. Neural Point Cloud Rendering via Multi-Plane Projection. In *CVPR*, 2020. 2, 4

[10] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured Light Fields. *Computer Graphics Forum*, 31, 2012. 2

[11] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshops*, 2019. 11

[12] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. DeepView: View Synthesis with Learned Gradient Descent. In *CVPR*, 2019. 3

[13] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. DeepStereo: Learning to Predict New Views from the World's Imagery. In *CVPR*, 2016. 3

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 4

[15] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep Blending for Free-Viewpoint Image-Based Rendering. In *SIGGRAPH Asia*, 2018. 2, 4

[16] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *SIGGRAPH Asia*, 35(6), 2016. 2

[17] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-Based View Synthesis for Light Field Cameras. *SIGGRAPH*, 35(6), 2016. 3

[18] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 5

[19] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *SIGGRAPH*, 36(4), 2017. 1, 2, 5, 6, 8, 11

[20] Johannes Kopf, Michael F Cohen, and Richard Szeliski. First-person Hyper-lapse Videos. *SIGGRAPH*, 33(4), 2014. 2

[21] Zhengqi Li, Wenqi Xian, Abe Davis, and Noah Snavely. Crowdsampling the Plenoptic Function. In *ECCV*, 2020. 3, 8

[22] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *SIGGRAPH*, 38(4), 2019. 3

[23] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. *arXiv:2008.02268*, 2020. 3, 8

[24] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *SIGGRAPH*, 2019. 3, 6, 7, 8

[25] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020. 3, 7, 8, 13

[26] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *CVPR*, 2020. 3

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019. 5, 11

[28] Eric Penner and Li Zhang. Soft 3D Reconstruction for View Synthesis. *SIGGRAPH*, 36(6), 2017. 2

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 4

[30] Gernot Riegler and Vladlen Koltun. Free View Synthesis. In *ECCV*, 2020. 1, 2, 4, 5, 6, 7, 8, 11, 12, 13

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015. 4

[32] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *CVPR*, 2016. 3, 11

[33] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise View Selection for Unstructured Multi-View Stereo. In *ECCV*, 2016. 3, 11

[34] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing*, 2000. 2

[35] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015. 5

[36] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *CVPR*, 2019. 3

[37] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *NeurIPS*, 2019. 3

[38] Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the Boundaries of View Extrapolation with Multiplane Images. In *CVPR*, 2019. 3

[39] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred Neural Rendering: Image Synthesis using Neural Textures. *SIGGRAPH*, 2019. 2, 4

[40] Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. Image-guided Neural Object Rendering. In *ICLR*, 2020. 2

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018. 4

[42] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image Quality Assessment: From Error Measurement to Structural Similarity. *TIP*, 13(4), 2004. 6

[43] Daniel N Wood, Daniel I Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H Salesin, and Werner Stuetzle. Surface Light Fields for 3D Photography. In *SIGGRAPH*, 2000. 2

[44] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep View Synthesis from Sparse Photometric Images. *SIGGRAPH*, 38(4), 2019. 3

[45] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492*, 2020. 3, 7, 8, 12

[46] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, 2018. 6

[47] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo Magnification: Learning view synthesis using multiplane images. *SIGGRAPH*, 37(4), 2018. 3

[48] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *SIGGRAPH*, 23(3), 2004. 3

8. Appendix

8.1. Algorithmic Overview and Implementation

Our Stable View Synthesis method has two major stages. In a first stage we set up the scene from a set of input images as outlined in Algorithm 1. This includes erecting the geometric scaffold and encoding all source images. In the second stage, we actually synthesize new images from novel target views. Given the preprocessed scene and a user-specified camera and viewpoint, we synthesize the new image as outlined in Algorithm 2.

Note that the steps in Algorithm 2 are easily parallelizable. For each pixel in the target view we backproject a single 3D point, which can be efficiently implemented as a single matrix-matrix multiplication to unproject all pixels. For the aggregation in 3D we first have to project each 3D point into each source image. If it projects to a valid image location, we bilinearly interpolate the feature vector. These operations are trivially parallelizable over the 3D points. The aggregation function itself can then be efficiently implemented with optimized gather operations as implemented in PyTorch Geometric [11]. The concatenation of 3D feature vectors and synthesizing an output image are implemented with standard functions and building blocks of the deep learning framework [27].

8.2. Evaluation Details

We train the scene-agnostic model with 600,000 iterations. The scene-specific model is trained with $256 \cdot N$ iterations starting from the scene-agnostic model, where N is the number of source images for the given scene.

The scene-agnostic model is trained on quarter-resolution images from the Tanks and Temples dataset [19]. To be comparable to prior work, we evaluate our method on half-resolution images. For Tanks and Temples [19] the output images are 980×546 pixels (slight variations for different scenes), for the FVS dataset [30] the images are 990×543 pixels (slight variations for different scenes), and on DTU the images are 800×600 pixels.

Algorithm 1 Scene Setup. Input is a set of source images $\{\mathcal{I}_n\}_{n=1}^N$ and the outputs are source camera parameters and viewpoints $\{\mathbf{K}_n, \mathbf{R}_n, \mathbf{t}_n\}_{n=1}^N$, the geometric scaffold Γ, and the encoded source images $\{\mathcal{F}_n\}_{n=1}^N$.

▷ *Erect geometric scaffold*

$\{\mathbf{K}_n, \mathbf{R}_n, \mathbf{t}_n\}_{n=1}^N = \text{structure-from-motion}(\{\mathcal{I}_n\}_{n=1}^N)$

$\Gamma = \text{surface-reconstruction}(\{\mathcal{I}_n, \mathbf{K}_n, \mathbf{R}_n, \mathbf{t}_n\}_{n=1}^N)$

▷ *Encode source*

for all $\mathcal{I}_n$ in $\{\mathcal{I}_n\}_{n=1}^N$ **do**

  $\mathcal{F}_n = \phi_{\text{enc}}(\mathcal{I}_n)$

end for

Algorithm 2 Stable View Synthesis. Input is the preprocessed scene $(\{\mathbf{K}_n, \mathbf{R}_n, \mathbf{t}_n\}_{n=1}^N, \Gamma, \{\mathcal{F}_n\}_{n=1}^N)$ and a target view defined by camera matrix $\mathbf{K}_t$ and pose $\mathbf{R}_t, \mathbf{t}_t$, output is an image $\mathcal{O}$ of the scene in the target view.

▷ *Get surface points*

$D = \text{render}(\Gamma, \mathbf{K}_t, \mathbf{R}_t, \mathbf{t}_t)$

$\mathcal{X} = \text{unproject}(D, \mathbf{K}_t, \mathbf{R}_t, \mathbf{t}_t)$

▷ *Aggregate feature vectors per 3D point*

for all $\mathbf{x}_{h,w}$ in $\mathcal{X}$ **do**

  $\mathbf{g}((\mathbf{x}, \mathbf{u})_{h,w}) = \phi_{\text{aggr}}(\mathbf{u}_{h,w}, \{(\mathbf{v}_k, \mathbf{f}_k(\mathbf{x}_{h,w}))\}_{k=1}^K)$

end for

▷ *Render Image*

$\mathcal{G} = [\mathbf{g}((\mathbf{x}, \mathbf{u})_{h,w})]_{h,w=1,1}^{H \times W}$

$O = \phi_{\text{render}}(\mathcal{G})$

8.3. Additional Results

We show qualitative results for the FVS dataset [30] in Figure 6. We observe that our method yields higher-fidelity results. As this dataset contains some frames that exhibit some motion blur, we noticed that our results are at times sharper than the ground truth.

Figure 7 shows qualitative extrapolation results for the DTU dataset [1]. Note that the ground-truth images have artifacts due to shadows from the camera setup (e.g., top of the skull). These artifacts are not visible in our synthesized images, which sometimes look better than the ground truth for this reason.

Please see the supplementary video for sequences.

8.4. Runtimes

In this section, we list the runtimes of our method and a selection of state-of-the-art methods. The numbers below are for a typical scene from the Tanks and Temples dataset [19].

We start with a breakdown of our method. We erect the geometric scaffold using COLMAP [32, 33]. Structure-from-motion takes <8 minutes (including feature extraction, feature matching, triangulation, and bundle adjustment), multi-view stereo takes <43 minutes, pointcloud fusion takes <14 minutes, and Delaunay-based surface reconstruction takes <32 minutes. This adds up to <97 minutes for erecting the geometric scaffold. We also encode all source images, which takes <25 seconds. Given a novel viewpoint, our method takes <1 second to synthesize an image. This can be sped up further, as our current implementation loads the encoded images from RAM to GPU memory for each novel target view. If the encoded images are already in GPU memory, image synthesis takes <0.2 seconds.

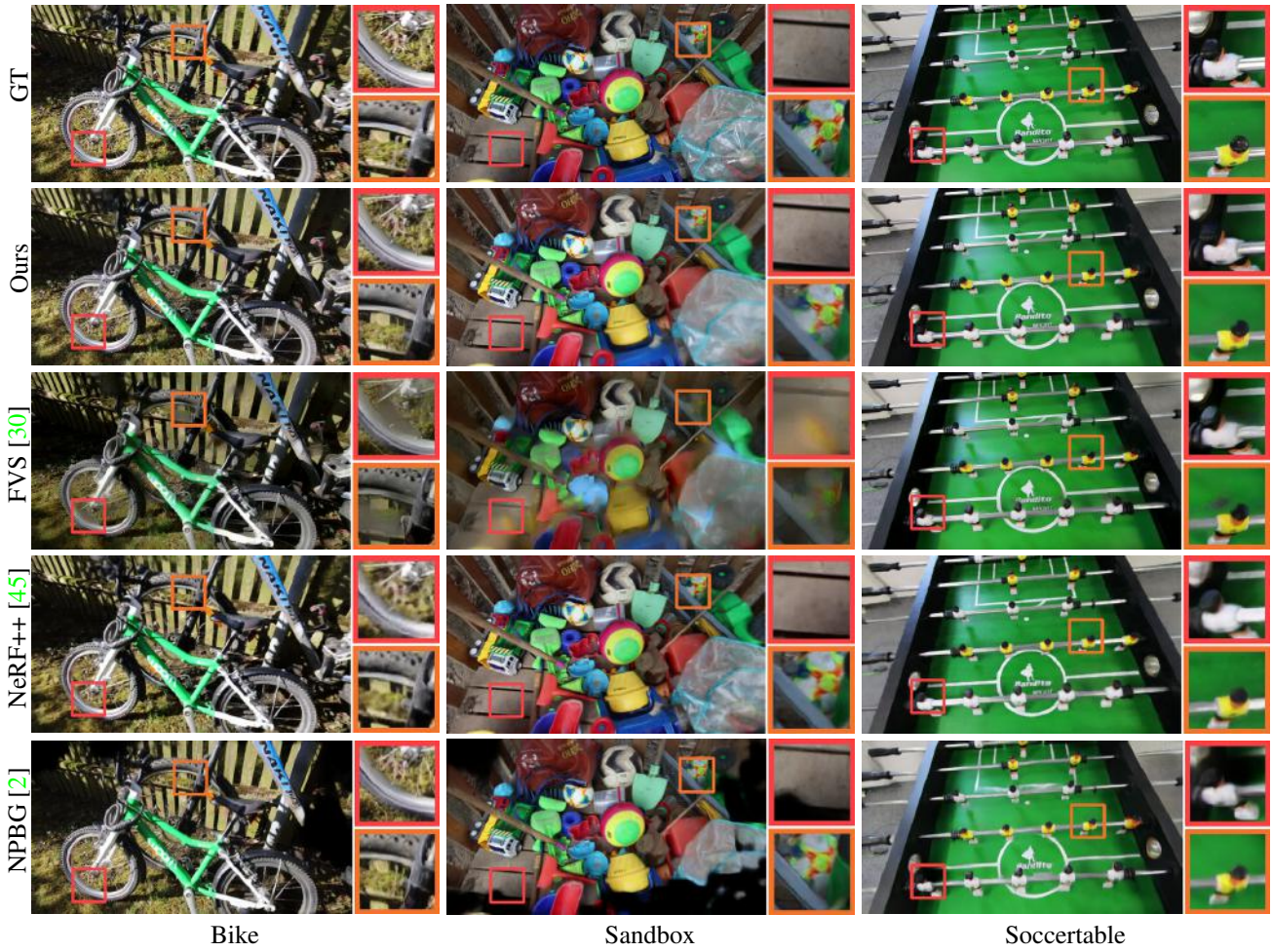


Figure 6: **Qualitative results on the FVS dataset.** Comparison of SVS to the best-performing prior methods.

NPBG [2] is based on a reconstructed point cloud of the scene. As outlined above, this can be computed in <65 minutes. Then, the NPBG representation has to be fitted to the scene. Starting from a pretrained rendering network, training for 10 epochs takes in total <31 minutes. As all feature vectors are kept in GPU memory, synthesizing novel views is fast, taking <0.1 seconds on average.

NeRF++ [45] requires less geometric information, only the camera poses and the sparse point cloud from structure-from-motion. As shown above, this can be computed in <8 minutes. Then, NeRF++ has to be fitted to the given scene. Optimizing it for 50,000 iterations takes <24 hours. To synthesize a novel target image from NeRF++ requires <71 seconds.

FVS [30] is based on the same geometric scaffold as our method, which can be erected in <97 minutes. Mapping 7 source images per novel target view and blending them via the recurrent network takes on average <0.5 seconds.

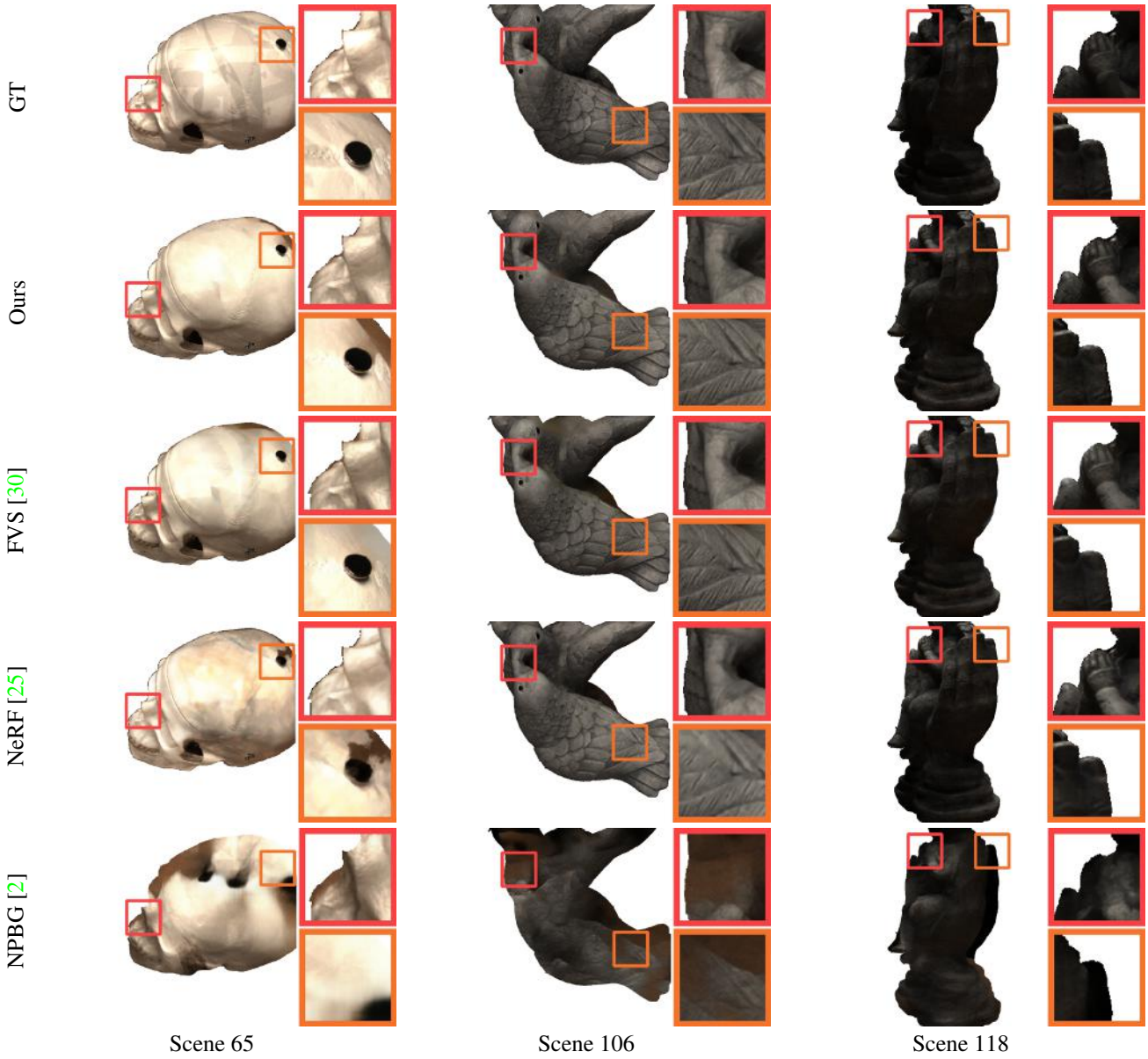


Figure 7: **Qualitative results on DTU.** Comparison of SVS to the best-performing prior methods.