A. Appendix

We provide a discussion of potential negative societal impact and include further details and results. Video results are available on the project website at srt-paper.github.io.

A.1. Potential Negative Societal Impact

Fakes. This paper proposes a method to synthesize novel views of scenes observed from a small set of input views. As in any image synthesis system, it could be used to produce images that have misleading or fake content, particularly if an adversary selected the input images. We plan to mitigate this issue partially by deploying the algorithms only in secure servers when running them on real world datasets, and we will clearly mark all synthesized imagery as such to reduce the chance of misinterpretation.

Privacy. This paper includes experiments run with a Street View dataset containing images acquired in public areas of real world outdoor scenes. As in any image dataset, these images could have information that negatively affects personal privacy. To mitigate these potential impacts, we follow strict privacy policies required by the owner of the data, Google, which include rules for using the data in accordance with regional laws, anonymizing the data to avoid identification of people or license plates, and retaining the data in secure storage (https://policies.google.com/privacy).

Fairness. Our Street View dataset contains images captured by a car-mounted camera platform driving on public roads in San Francisco. Long sequences of images are captured systematically with fixed view directions and regular time intervals as the car drives. There are biases in the dataset based on which roads are driven in which city, which align with demographic or economic factors. However, individual images are not selected by a human photographer, which might mitigate some types of bias common in other computer vision datasets.

Energy consumption. The paper proposes a method to train a transformer network architecture to produce a scene representation and arbitrary novel images from a set of input images. The training procedure requires a lot of compute cycles, and thus has a negative impact on the environment due to its energy consumption. However, in comparison to most competing methods based on volumetric rendering methods, the inference cost is smaller (by 6-7 orders of magnitude per scene and 2-3 orders of magnitude per image). Thus, our proposed system is the most efficient of its kind when deployed at scale (our target use case) – where the system is trained on millions of images and then run without further fine-tuning to synthesize up to trillions of novel images.

A.2. Architecture, Training and Evaluation

Dataset Details

Neural 3D Mesh Renderer Dataset (NMR) [17]. This dataset consists of ShapeNet [4] objects from the 13 largest classes, each with 24 fixed views rendered at 64×64 pixels. We use the SoftRas split [14], which contains 31 k training, 4 k validation and 9 k test objects. Following the protocol in [38, 49], one randomly sampled image is provided as input, and images are rendered for all other 23 views.

MultiShapeNet (MSN). We use 1 M scenes for training, and 10 k for testing. To ensure that all test set scenes contain novel objects, we split the full set of ShapeNet objects into a train and test split, and sample objects for each from the corresponding split. Viewpoints are selected by independently and uniformly sampling 10 cameras within a half-sphere shell of radius 8-12 for each scene, with all cameras pointed at the origin. All images are rendered at a resolution of 128×128. The task is to render novel views given 5 randomly selected input images of a novel scene.

Street View. The resolution of this dataset is 176×128 pixels. During training, the task is to predict 5 randomly sampled images from the other 5 input views. At test time, we render views from novel viewpoints. To ensure that the evaluation scenes are new to the model, we physically separate them from the training scenes.

SRT Model Details

We use the same network architecture and hyper parameters throughout all experiments (with the exception of the appearance encoding module, see Sec. A.2). Our CNN consists of 4 blocks, each with 2 convolutional layers. The first convolution in each block has stride 1, the second has stride 2. We begin with 96 channels which we double with every strided convolution, and map the final activations with a 1×1 convolution (*i.e.*, a per-patch linear layer) to 768 channels.

Both encoder and decoder transformers are similar to ViT-Base [6]: all attention layers have 12 heads, each with 64 channels, and the MLPs have 1 hidden layer with 1536 channels and GELU [13] activations. The final MLP for the decoder also has a single hidden layer, but with 128 channels, with a final sigmoid operation for the RGB output. For the positional encoding of ray origin and direction, we use $L_{\mathbf{o}} = L_{\mathbf{d}} = 15$.

We train our model with the Adam optimizer [19] with initial learning rate 1×10^{-4} that is smoothly decayed down to 1.6×10^{-5} over 4 M training steps and a learning rate warmup phase in the first 2.5 k steps. We train with a batch size of 256 and for every data point, we randomly sample 8192 points uniformly across all target views for rendering and the loss computation. We noticed that the model was still not fully converged at 4 M training steps, and validation

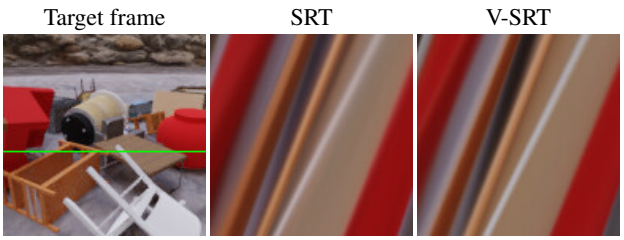

Figure 8. **Epipolar Images (EPI)** – For a dolly camera motion, the same horizontal line of pixels marked in green (left) is stacked vertically from all video frames, producing an EPI. The slope of each diagonal line in an EPI corresponds to the depth of that pixel in the scene. V-SRT uses a volumetric parametrization of the scene and no viewing directions. We can see that its EPI looks quite similar to SRT's, indicating that the latter's scene representation is geometrically and temporally stable, despite a lack of theoretical consistency guarantees for our light field parametrization.

metrics often keep improving up to and beyond 10 M, the maximum to which we trained the model (especially on more challenging datasets), though qualitative improvements are increasingly subtle as training progresses.

Baseline Details

LFN. We received the LFN [38] code through private communications with the authors. The code was adapted to read the MSN dataset. We trained LFN for 500 k steps on the NMR dataset with batch size 128 and using the Adam optimizer with learning rate 1×10^{-4}. After training, the weights were frozen, and only the latent codes were optimized for 2 k epochs with batch size 64. For MSN, we trained LFN using a batch size of 16 for 500 k steps as training converged much faster and to lower loss values than with the default batch size of 128. For the camera pose noise ablations, we trained for 350 k steps (those models converged earlier) with batch size 16. After training, we performed latent code optimization for 200 k steps using batch size 16.

PixelNeRF. We used the official implementation from the authors [2]. PixelNerf [49] was trained for 1 M steps with the configuration from the codebase used for all experiments in the original paper. During training, each batch contains 4 scenes of 10 images each for the MSN dataset, and 128 rays are sampled per scene for the loss. Training uses Adam with learning rate 1×10^{-4}. In consultation with the authors, we applied a minor fix in the projection code, tweaking a division operation to avoid NaNs in some experiments.

Appearance Encoding

As the Street View data contains variations in exposure and white balance, we extend our model with an appearance encoder for these experiments. During training, we extract an appearance embedding from the respective target views and concatenate it to the input of the final 2-layer MLP after the decoder transformer, see Fig. 2. The appearance encoder consists of 4 blocks of 2×2 mean pooling operations followed by 1x1 convolutions with 32 channels, and a ReLU activation. Finally, we take the mean over the spatial dimensions and map the vector to 4 channels before passing it to the decoder MLP. At test time, the appearance embedding of one of the input images can be used. Fig. 9 shows the effect of the appearance embedding on the output of the model.

Figure 9. **Appearance embeddings** – In some scenes, white balance and exposure are not consistent among the input views, see the shift in colors between the reference views in the top row. We add a simple appearance encoder to SRT to handle these cases. Note that the appearance embedding only changes the white balance of the rendered images without changing the structure (bottom row). Further, it generalizes across entirely different views of the scene (*e.g.*, leftmost column). The effect is similar to the appearance handling in [23], however, we do not need an expensive optimization procedure to get the appearance embedding of an image, it is a very fast feed-forward pass, and we do not need hundreds to thousands of images of the same scene to capture appearance variations.

Semantic Segmentation

We train the semantics decoder $\mathcal{D}_{\text{sem}}$ by minimizing the cross-entropy between softmax class predictions and ground truth class distributions p for each ray / pixel:

$$\arg\min_{\theta_{\mathcal{D}_{\text{sem}}}} \; -\sum_{s} \mathbb{E}_{\mathbf{r}\sim\mathbf{I}^{\text{gt}}_{s,i}} p(\mathbf{r}) \cdot \log(\mathcal{D}_{\text{sem}}(\mathbf{r})) \qquad (10)$$

The ground truth classes were generated by running a pretrained 2D semantic segmentation model on the training views, and extracting one-hot predictions.

Benchmark Details

We measure wall-clock performance of all models on a single NVIDIA V100. In Tab. 3, we measure three different settings. For A), we measure the onboarding time of a novel scene, *i.e.* for PixelNeRF and SRT, this is the execution time of the encoder, while for LFN, it is the time to optimize the scene-latent. Hence, LFN takes minutes while PixelNeRF and SRT encode the scene in 5-10 ms. For B), once the scene has

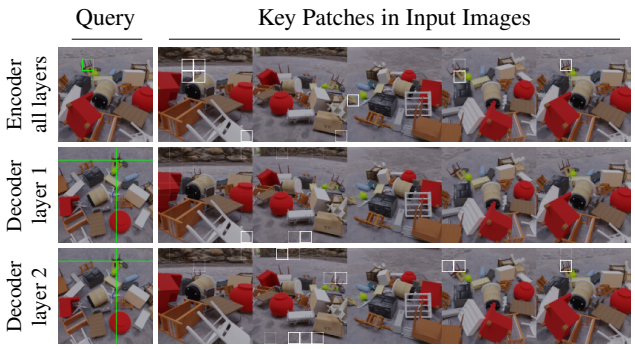

Figure 10. **Attention visualization for UpSRT** – The attention patterns of UpSRT resemble the ones of SRT (Fig. 5). Notably, the model attends into relevant parts of *all* inputs, showing that UpSRT has learned to use unposed imagery even in complex scenes.

already been onboarded, we measure the rendering FPS in a streaming / interactive setup, *i.e*. frame by frame, without batching. Finally, C) measures a full application scenario that combines scene encoding and video rendering. Here, we measure the full time taken to onboard a new scene and render a 100-frame video. Note that we allow batching in this scenario, as we assume the camera render path is known, hence the measured times are significantly lower than the sum of scene encoding and single-image rendering times.

A.3. Experiments and Results

UpSRT Model Inspection

UpSRT works without input poses at test time – however, the pose of the first camera is always known by definition, as all target poses are given relative to its reference frame (see Sec. 3). Note that this does *not* mean UpSRT needs poses – target views must always be defined somehow, and in this case, they are simply defined relative to an arbitrary input view. Hence, the fact that UpSRT works and renders meaningful images by itself does not prove yet that the model is capable of using unposed imagery; the model might only be using the first input camera and ignoring all other input views. In the following, we show that this is not the case, and that UpSRT indeed uses all input cameras.

First, we investigate the attention patterns for UpSRT in Fig. 10. As the results show, UpSRT is using all input images for reconstructing novel views of the scene, showing that it is capable of finding similar structures purely based on imagery, and make use of that information for more detailed scene reconstructions. Note that in the encoder, patches in other input images attend strongly specifically into the reference camera, possibly to *spread* pose information to all other input views. That could also explain why the decoder does not attend strongly into the reference camera.

Second, we compare UpSRT trained and tested on 5 input images with SRT trained and tested only on one input image. If UpSRT was only trivially using the first (practically posed) input image, the quality of these two models should be similar. However, we see that SRT only achieves a PSNR / SSIM / LPIPS of 19.2 / 0.53 / 0.52 on MSN in this setup, while UpSRT reaches much better values of 22.2 / 0.65 / 0.41, respectively.

Finally, we take a closer look at the first row in Fig. 4 and notice that the red car is *not* visible in the first input image (leftmost column), but in other, unposed input views. Despite this, UpSRT is correctly reconstructing the red car at the correct 3D position, showing that the model has detected the position of objects in unposed imagery and has correctly integrated them into the scene representation. Fig. 14 shows further qualitative results for UpSRT.

V-SRT Volumetric Rendering

We show results for V-SRT in Fig. 11. As can be seen, the render quality is comparable to that of SRT (see Fig. 7), with the addition that depth maps can now be acquired as well. Similarly, Fig. 14 shows results on the MSN dataset.

Robustness Study

Fig. 13 visualizes the effect of camera noise in our robustness studies, see Sec. 4.4. As can be seen, the amount of noise is subtle, yet enough to severely affect reconstruction quality of geometry-based methods as seen in Fig. 6.

Temporal Consistency

Fig. 8 shows EPI for a dolly motion on an MSN scene. The results show that SRT learns a coherent 3D scene. Note that this consistency also implies that depth maps could be extracted from our model [34, 47, 45], see also [38].

Comparison to NeRF-based Methods

In contrast to SRT and the baselines we compare to in the paper, NeRF needs costly per-scene optimization, and is known to fail in the few-image setting. In Fig. 12, we compare SRT with Mip-NeRF [3] (improved version of NeRF) and DietNeRF [15] (specialized to handle few images), and see that they both fail for this highly challenging dataset with only 5 images.

Figure 11. **V-SRT results on Street View** – V-SRT has similar quality to SRT (see Fig. 7), but is much slower at inference due to the sampling procedure for volumetric rendering. However, it allows one to render depth maps, which look reasonable in most scenes. In the center example, the depth of the car is inaccurate as a result of the lighting variation, note that *e.g.* the car changes color due to the metallic surface (*e.g.*, input views 2 *vs*. 3), and since we did not include the viewing direction (see Sec. 4.3), the model has learned to account for this in the geometry instead.

Figure 12. Per-scene, NeRF-based optimization methods compared to SRT on the Street View dataset. Both Mip-NeRF [3] and DietNeRF [15] need costly per-scene optimization, and fail with only 5 input images.

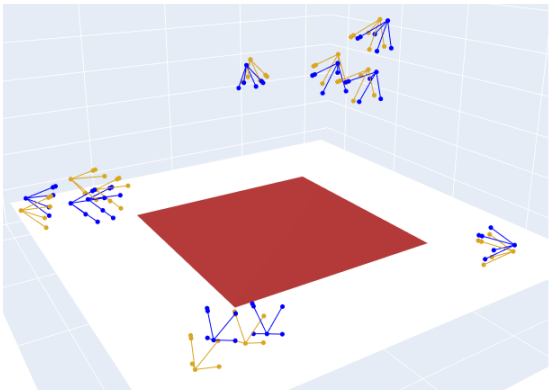

Figure 13. **Camera noise** – Visualization of the effect of camera noise for $\sigma = 0.1$, see Sec. 4.4. *Blue*: original cameras, *brown*: cameras with noise, *red*: scene bounding box containing all objects.

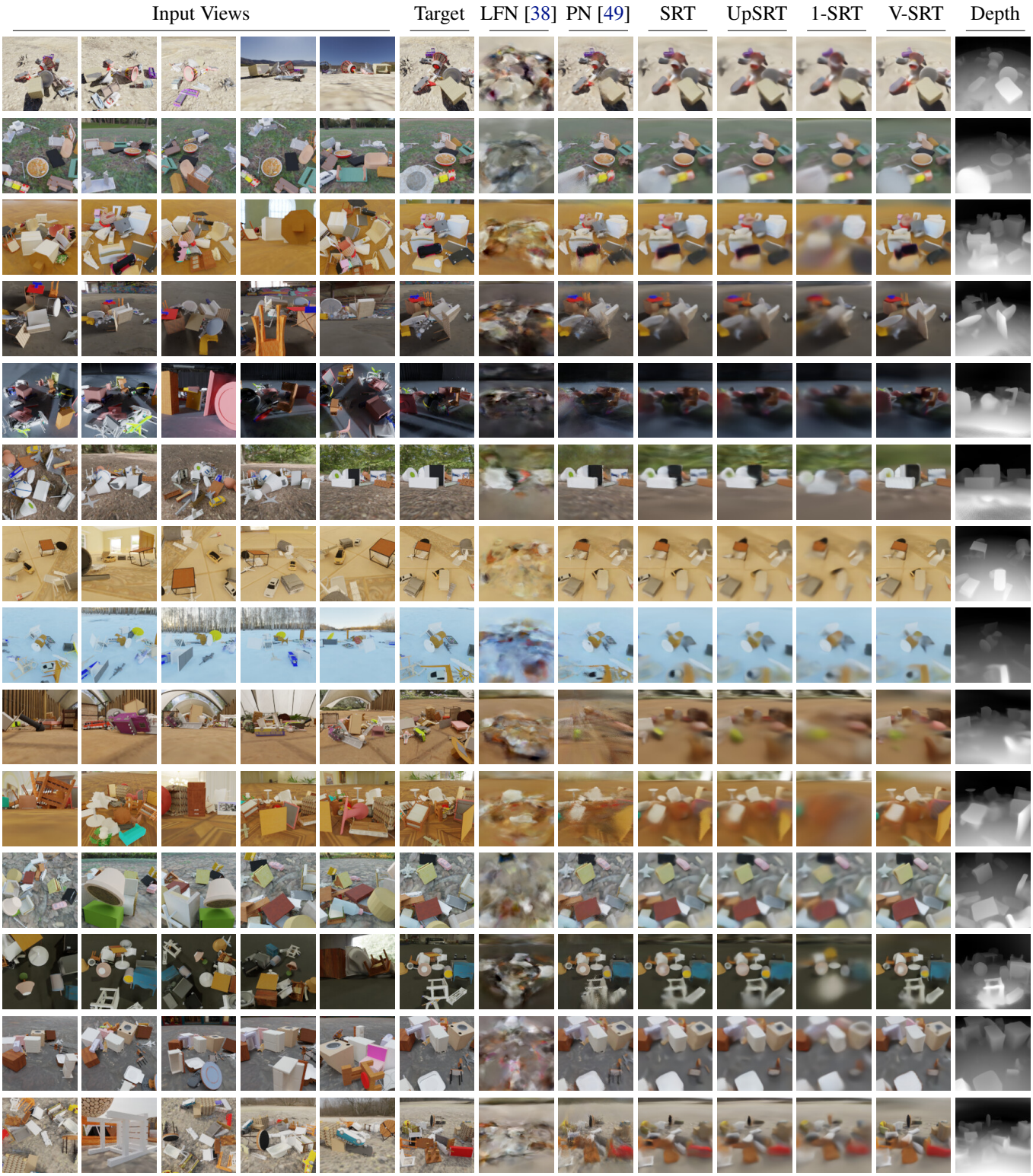

Figure 14. **Qualitative results on MultiShapeNet** – More results on the MSN dataset from LFN [38], PixelNeRF [49], and SRT including variations with unposed inputs (UpSRT), only the leftmost input image (1-SRT), and volumetric rendering (V-SRT & depth).