

---

# Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations

---

**Vincent Sitzmann   Michael Zollhöfer   Gordon Wetzstein**

{sitzmann, zollhoefer}@cs.stanford.edu, gordon.wetzstein@stanford.edu  
Stanford University

[vsitzmann.github.io/srns/](https://vsitzmann.github.io/srns/)

## Abstract

Unsupervised learning with generative models has the potential of discovering rich representations of 3D scenes. While geometric deep learning has explored 3D-structure-aware representations of scene geometry, these models typically require explicit 3D supervision. Emerging neural scene representations can be trained only with posed 2D images, but existing methods ignore the three-dimensional structure of scenes. We propose Scene Representation Networks (SRNs), a continuous, 3D-structure-aware scene representation that encodes both geometry and appearance. SRNs represent scenes as continuous functions that map world coordinates to a feature representation of local scene properties. By formulating the image formation as a differentiable ray-marching algorithm, SRNs can be trained end-to-end from only 2D images and their camera poses, without access to depth or shape. This formulation naturally generalizes across scenes, learning powerful geometry and appearance priors in the process. We demonstrate the potential of SRNs by evaluating them for novel view synthesis, few-shot reconstruction, joint shape and appearance interpolation, and unsupervised discovery of a non-rigid face model.<sup>1</sup>

## 1 Introduction

A major driver behind recent work on generative models has been the promise of unsupervised discovery of powerful neural scene representations, enabling downstream tasks ranging from robotic manipulation and few-shot 3D reconstruction to navigation. A key aspect of solving these tasks is understanding the three-dimensional structure of an environment. However, prior work on neural scene representations either does not or only weakly enforces 3D structure [1–4]. Multi-view geometry and projection operations are performed by a black-box neural renderer, which is expected to learn these operations from data. As a result, such approaches fail to discover 3D structure under limited training data (see Sec. 4), lack guarantees on multi-view consistency of the rendered images, and learned representations are generally not interpretable. Furthermore, these approaches lack an intuitive interface to multi-view and projective geometry important in computer graphics, and cannot easily generalize to camera intrinsic matrices and transformations that were completely unseen at training time.

In geometric deep learning, many classic 3D scene representations, such as voxel grids [5–10], point clouds [11–14], or meshes [15] have been integrated with end-to-end deep learning models and have led to significant progress in 3D scene understanding. However, these scene representations are discrete, limiting achievable spatial resolution, only sparsely sampling the underlying smooth surfaces of a scene, and often require explicit 3D supervision.

---

<sup>1</sup>Please see supplemental video for additional results.

We introduce *Scene Representation Networks* (SRNs), a continuous neural scene representation, along with a differentiable rendering algorithm, that model both 3D scene geometry and appearance, enforce 3D structure in a multi-view consistent manner, and naturally allow generalization of shape and appearance priors across scenes. The key idea of SRNs is to represent a scene implicitly as a continuous, differentiable function that maps a 3D world coordinate to a feature-based representation of the scene properties at that coordinate. This allows SRNs to naturally interface with established techniques of multi-view and projective geometry while operating at high spatial resolution in a memory-efficient manner. SRNs can be trained end-to-end, supervised only by a set of posed 2D images of a scene. SRNs generate high-quality images *without any 2D convolutions*, exclusively operating on individual pixels, which enables image generation at arbitrary resolutions. They generalize naturally to camera transformations and intrinsic parameters that were completely unseen at training time. For instance, SRNs that have only ever seen objects from a constant distance are capable of rendering close-ups of said objects flawlessly. We evaluate SRNs on a variety of challenging 3D computer vision problems, including novel view synthesis, few-shot scene reconstruction, joint shape and appearance interpolation, and unsupervised discovery of a non-rigid face model.

To summarize, our approach makes the following key contributions:

- A continuous, 3D-structure-aware neural scene representation and renderer, SRNs, that efficiently encapsulate both scene geometry and appearance.
- End-to-end training of SRNs without explicit supervision in 3D space, purely from a set of posed 2D images.
- We demonstrate novel view synthesis, shape and appearance interpolation, and few-shot reconstruction, as well as unsupervised discovery of a non-rigid face model, and significantly outperform baselines from recent literature.

**Scope** The current formulation of SRNs does not model view- and lighting-dependent effects or translucency, reconstructs shape and appearance in an entangled manner, and is non-probabilistic. Please see Sec. 5 for a discussion of future work in these directions.

## 2 Related Work

Our approach lies at the intersection of multiple fields. In the following, we review related work.

**Geometric Deep Learning.** Geometric deep learning has explored various representations to reason about scene geometry. Discretization-based techniques use voxel grids [7, 16–22], octree hierarchies [23–25], point clouds [11, 26, 27], multiplane images [28], patches [29], or meshes [15, 21, 30, 31]. Methods based on function spaces continuously represent space as the decision boundary of a learned binary classifier [32] or a continuous signed distance field [33–35]. While these techniques are successful at modeling geometry, they often require 3D supervision, and it is unclear how to efficiently infer and represent appearance. Our proposed method encapsulates both scene geometry and appearance, and can be trained end-to-end via learned differentiable rendering, supervised only with posed 2D images.

**Neural Scene Representations.** Latent codes of autoencoders may be interpreted as a feature representation of the encoded scene. Novel views may be rendered by concatenating target pose and latent code [1] or performing view transformations directly in the latent space [4]. Generative Query Networks [2, 3] introduce a probabilistic reasoning framework that models uncertainty due to incomplete observations, but both the scene representation and the renderer are oblivious to the scene’s 3D structure. Some prior work infers voxel grid representations of 3D scenes from images [6, 8, 9] or uses them for 3D-structure-aware generative models [10, 36]. Graph neural networks may similarly capture 3D structure [37]. Compositional structure may be modeled by representing scenes as programs [38]. We demonstrate that models with scene representations that ignore 3D structure fail to perform viewpoint transformations in a regime of limited (but significant) data, such as the Shapenet v2 dataset [39]. Instead of a discrete representation, which limits achievable spatial resolution and does not smoothly parameterize scene surfaces, we propose a continuous scene representation.

**Neural Image Synthesis.** Deep models for 2D image and video synthesis have recently shown promising results in generating photorealistic images. Some of these approaches are based on

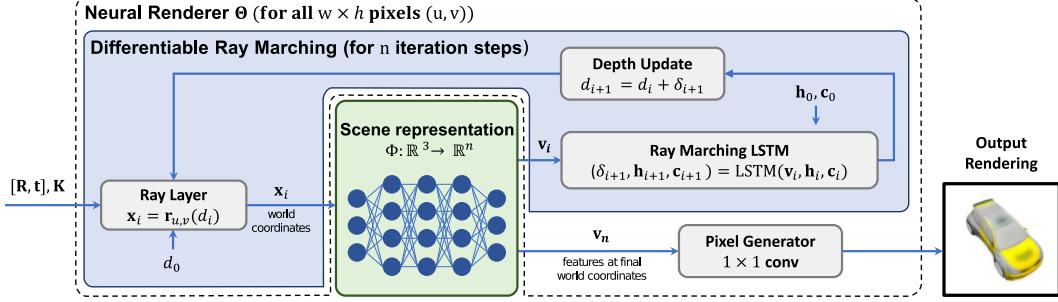


Figure 1: Overview: at the heart of SRNs lies a continuous, 3D-aware neural scene representation,  $\Phi$ , which represents a scene as a function that maps  $(x, y, z)$  world coordinates to a feature representation of the scene at those coordinates (see Sec. 3.1). A neural renderer  $\Theta$ , consisting of a learned ray marcher and a pixel generator, can render the scene from arbitrary novel view points (see Sec. 3.2).

(variational) auto-encoders [40, 41], generative flows [42, 43], or autoregressive per-pixel models [44, 45]. In particular, generative adversarial networks [46–50] and their conditional variants [51–53] have recently achieved photo-realistic single-image generation. Compositional Pattern Producing Networks [54, 55] learn functions that map 2D image coordinates to color. Some approaches build on explicit spatial or perspective transformations in the networks [56–58, 14]. Recently, following the spirit of “vision as inverse graphics” [59, 60], deep neural networks have been applied to the task of inverting graphics engines [61–65]. However, these 2D generative models only learn to parameterize the manifold of 2D natural images, and struggle to generate images that are multi-view consistent, since the underlying 3D scene structure cannot be exploited.

### 3 Formulation

Given a training set  $\mathcal{C} = \{(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i)\}_{i=1}^N$  of  $N$  tuples of images  $\mathcal{I}_i \in \mathbb{R}^{H \times W \times 3}$  along with their respective extrinsic  $\mathbf{E}_i = [\mathbf{R} | \mathbf{t}] \in \mathbb{R}^{3 \times 4}$  and intrinsic  $\mathbf{K}_i \in \mathbb{R}^{3 \times 3}$  camera matrices [66], our goal is to distill this dataset of observations into a neural scene representation  $\Phi$  that strictly enforces 3D structure and allows to generalize shape and appearance priors across scenes. In addition, we are interested in a rendering function  $\Theta$  that allows us to render the scene represented by  $\Phi$  from arbitrary viewpoints. In the following, we first formalize  $\Phi$  and  $\Theta$  and then discuss a framework for optimizing  $\Phi, \Theta$  for a single scene given only posed 2D images. Note that this approach does *not* require information about scene geometry. Additionally, we show how to learn a family of scene representations for an entire class of scenes, discovering powerful shape and appearance priors.

#### 3.1 Representing Scenes as Functions

Our key idea is to represent a scene as a function  $\Phi$  that maps a spatial location  $\mathbf{x}$  to a feature representation  $\mathbf{v}$  of learned scene properties at that spatial location:

$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^n, \quad \mathbf{x} \mapsto \Phi(\mathbf{x}) = \mathbf{v}. \quad (1)$$

The feature vector  $\mathbf{v}$  may encode visual information such as surface color or reflectance, but it may also encode higher-order information, such as the signed distance of  $\mathbf{x}$  to the closest scene surface. This continuous formulation can be interpreted as a generalization of discrete neural scene representations. Voxel grids, for instance, discretize  $\mathbb{R}^3$  and store features in the resulting 3D grid [5–10]. Point clouds [12–14] may contain points at any position in  $\mathbb{R}^3$ , but only sparsely sample surface properties of a scene. In contrast,  $\Phi$  densely models scene properties and can in theory model arbitrary spatial resolutions, as it is continuous over  $\mathbb{R}^3$  and can be sampled with arbitrary resolution. In practice, we represent  $\Phi$  as a multi-layer perceptron (MLP), and spatial resolution is thus limited by the capacity of the MLP.

In contrast to recent work on representing scenes as unstructured or weakly structured feature embeddings [1, 4, 2],  $\Phi$  is explicitly aware of the 3D structure of scenes, as the input to  $\Phi$  are world coordinates  $(x, y, z) \in \mathbb{R}^3$ . This allows interacting with  $\Phi$  via the toolbox of multi-view and perspective geometry that the physical world obeys, only using learning to approximate the unknown properties of the scene itself. In Sec. 4, we show that this formulation leads to multi-view consistent novel view synthesis, data-efficient training, and a significant gain in model interpretability.

### 3.2 Neural Rendering

Given a scene representation  $\Phi$ , we introduce a neural rendering algorithm  $\Theta$ , that maps a scene representation  $\Phi$  as well as the intrinsic  $\mathbf{K}$  and extrinsic  $\mathbf{E}$  camera parameters to an image  $\mathcal{I}$ :

$$\Theta : \mathcal{X} \times \mathbb{R}^{3 \times 4} \times \mathbb{R}^{3 \times 3} \rightarrow \mathbb{R}^{H \times W \times 3}, \quad (\Phi, \mathbf{E}, \mathbf{K}) \mapsto \Theta(\Phi, \mathbf{E}, \mathbf{K}) = \mathcal{I}, \quad (2)$$

where  $\mathcal{X}$  is the space of all functions  $\Phi$ .

The key complication in rendering a scene represented by  $\Phi$  is that geometry is represented implicitly. The surface of a wooden table top, for instance, is defined by the subspace of  $\mathbb{R}^3$  where  $\Phi$  undergoes a change from a feature vector representing free space to one representing wood.

To render a single pixel in the image observed by a virtual camera, we thus have to solve two sub-problems: (i) finding the world coordinates of the intersections of the respective camera rays with scene geometry, and (ii) mapping the feature vector  $\mathbf{v}$  at that spatial coordinate to a color. We will first propose a neural ray marching algorithm with learned, adaptive step size to find ray intersections with scene geometry, and subsequently discuss the architecture of the pixel generator network that learns the feature-to-color mapping.

#### 3.2.1 Differentiable Ray Marching Algorithm

---

##### Algorithm 1 Differentiable Ray-Marching

---

```

1: function FINDINTERSECTION( $\Phi, \mathbf{K}, \mathbf{E}, (u, v)$ )
2:    $d_0 \leftarrow 0.05$                                       $\triangleright$  Near plane
3:    $(\mathbf{h}_0, \mathbf{c}_0) \leftarrow (\mathbf{0}, \mathbf{0})$             $\triangleright$  Initial state of LSTM
4:   for  $i \leftarrow 0$  to  $max\_iter$  do
5:      $\mathbf{x}_i \leftarrow \mathbf{r}_{u,v}(d_i)$                        $\triangleright$  Calculate world coordinates
6:      $\mathbf{v}_i \leftarrow \Phi(\mathbf{x}_i)$                           $\triangleright$  Extract feature vector
7:      $(\delta, \mathbf{h}_{i+1}, \mathbf{c}_{i+1}) \leftarrow LSTM(\mathbf{v}_i, \mathbf{h}_i, \mathbf{c}_i)$   $\triangleright$  Predict steplength using ray marching LSTM
8:      $d_{i+1} \leftarrow d_i + \delta$                             $\triangleright$  Update d
9:   return  $\mathbf{r}_{u,v}(d_{max\_iter})$ 

```

---

Intersection testing intuitively amounts to solving an optimization problem, where the point along each camera ray is sought that minimizes the distance to the surface of the scene. To model this problem, we parameterize the points along each ray, identified with the coordinates  $(u, v)$  of the respective pixel, with their distance  $d$  to the camera ( $d > 0$  represents points in front of the camera):

$$\mathbf{r}_{u,v}(d) = \mathbf{R}^T(\mathbf{K}^{-1} \begin{pmatrix} u \\ v \\ d \end{pmatrix} - \mathbf{t}), \quad d > 0, \quad (3)$$

with world coordinates  $\mathbf{r}_{u,v}(d)$  of a point along the ray with distance  $d$  to the camera, camera intrinsics  $\mathbf{K}$ , and camera rotation matrix  $\mathbf{R}$  and translation vector  $\mathbf{t}$ . For each ray, we aim to solve

$$\begin{aligned} \arg \min \quad & d \\ \text{s.t.} \quad & \mathbf{r}_{u,v}(d) \in \Omega, \quad d > 0 \end{aligned} \quad (4)$$

where we define the set of all points that lie on the surface of the scene as  $\Omega$ .

Here, we take inspiration from the classic sphere tracing algorithm [67]. Sphere tracing belongs to the class of ray marching algorithms, which solve Eq. 4 by starting at a distance  $d_{init}$  close to the camera and stepping along the ray until scene geometry is intersected. Sphere tracing is defined by a special choice of the step length: each step has a length equal to the signed distance to the closest surface point of the scene. Since this distance is only 0 on the surface of the scene, the algorithm takes non-zero steps until it has arrived at the surface, at which point no further steps are taken. Extensions of this algorithm propose heuristics to modifying the step length to speed up convergence [68]. We instead propose to *learn* the length of each step.

Specifically, we introduce a *ray marching long short-term memory (RM-LSTM)* [69], that maps the feature vector  $\Phi(\mathbf{x}_i) = \mathbf{v}_i$  at the current estimate of the ray intersection  $\mathbf{x}_i$  to the length of the next ray marching step. The algorithm is formalized in Alg. 1.

Given our current estimate  $d_i$ , we compute world coordinates  $\mathbf{x}_i = \mathbf{r}_{u,v}(d_i)$  via Eq. 3. We then compute  $\Phi(\mathbf{x}_i)$  to obtain a feature vector  $\mathbf{v}_i$ , which we expect to encode information about nearby scene surfaces. We then compute the step length  $\delta$  via the RM-LSTM as  $(\delta, \mathbf{h}_{i+1}, \mathbf{c}_{i+1}) = LSTM(\mathbf{v}_i, \mathbf{h}_i, \mathbf{c}_i)$ , where  $\mathbf{h}$  and  $\mathbf{c}$  are the output and cell states, and increment  $d_i$  accordingly. We iterate this process for a constant number of steps. This is critical, because a dynamic termination criterion would have no guarantee for convergence in the beginning of the training, where both  $\Phi$  and the ray marching LSTM are initialized at random. The final step yields our estimate of the world coordinates of the intersection of the ray with scene geometry. The  $z$ -coordinates of running and final estimates of intersections in camera coordinates yield depth maps, which we denote as  $\mathbf{d}_i$ , which visualize every step of the ray marcher. This makes the ray marcher interpretable, as failures in geometry estimation show as inconsistencies in the depth map. Note that depth maps are differentiable with respect to all model parameters, but are not required for training  $\Phi$ . Please see the supplement for a contextualization of the proposed rendering approach with classical rendering algorithms.

### 3.2.2 Pixel Generator Architecture

The pixel generator takes as input the 2D feature map sampled from  $\Phi$  at world coordinates of ray-surface intersections and maps it to an estimate of the observed image. As a generator architecture, we choose a per-pixel MLP that maps a single feature vector  $\mathbf{v}$  to a single RGB vector. This is equivalent to a convolutional neural network (CNN) with only  $1 \times 1$  convolutions. Formulating the generator without 2D convolutions has several benefits. First, the generator will always map the same  $(x, y, z)$  coordinate to the same color value. Assuming that the ray-marching algorithm finds the correct intersection, the rendering is thus trivially multi-view consistent. This is in contrast to 2D convolutions, where the value of a single pixel depends on a neighborhood of features in the input feature map. When transforming the camera in 3D, e.g. by moving it closer to a surface, the 2D neighborhood of a feature may change. As a result, 2D convolutions come with no guarantee on multi-view consistency. With our per-pixel formulation, the rendering function  $\Theta$  operates independently on all pixels, allowing images to be generated with arbitrary resolutions and poses. On the flip side, we cannot exploit recent architectural progress in CNNs, and a per-pixel formulation requires the ray marching, the SRNs and the pixel generator to operate on the same (potentially high) resolution, requiring a significant memory budget. Please see the supplement for a discussion of this trade-off.

## 3.3 Generalizing Across Scenes

We now generalize SRNs from learning to represent a single scene to learning shape and appearance priors over several instances of a single class. Formally, we assume that we are given a set of  $M$  instance datasets  $\mathcal{D} = \{\mathcal{C}_j\}_{j=1}^M$ , where each  $\mathcal{C}_j$  consists of tuples  $\{(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i)\}_{i=1}^N$  as discussed in Sec. 3.1.

We reason about the set of functions  $\{\Phi_j\}_{j=1}^M$  that represent instances of objects belonging to the same class. By parameterizing a specific  $\Phi_j$  as an MLP, we can represent it with its vector of parameters  $\phi_j \in \mathbb{R}^l$ . We assume scenes of the same class have common shape and appearance properties that can be fully characterized by a set of latent variables  $\mathbf{z} \in \mathbb{R}^k$ ,  $k < l$ . Equivalently, this assumes that all parameters  $\phi_j$  live in a  $k$ -dimensional subspace of  $\mathbb{R}^l$ . Finally, we define a mapping

$$\Psi : \mathbb{R}^k \rightarrow \mathbb{R}^l, \quad \mathbf{z}_j \mapsto \Psi(\mathbf{z}_j) = \phi_j \tag{5}$$

that maps a latent vector  $\mathbf{z}_j$  to the parameters  $\phi_j$  of the corresponding  $\Phi_j$ . We propose to parameterize  $\Psi$  as an MLP, with parameters  $\psi$ . This architecture was previously introduced as a Hypernetwork [70], a neural network that regresses the parameters of another neural network. We share the parameters of the rendering function  $\Theta$  across scenes. We note that assuming a low-dimensional embedding manifold has so far mainly been empirically demonstrated for classes of single objects. Here, we similarly only demonstrate generalization over classes of single objects.

**Finding latent codes  $\mathbf{z}_j$ .** To find the latent code vectors  $\mathbf{z}_j$ , we follow an auto-decoder framework [33]. For this purpose, each object instance  $\mathcal{C}_j$  is represented by its own latent code  $\mathbf{z}_j$ . The  $\mathbf{z}_j$  are free variables and are optimized jointly with the parameters of the hypernetwork  $\Psi$  and the neural renderer  $\Theta$ . We assume that the prior distribution over the  $\mathbf{z}_j$  is a zero-mean multivariate Gaussian with a diagonal covariance matrix. Please refer to [33] for additional details.

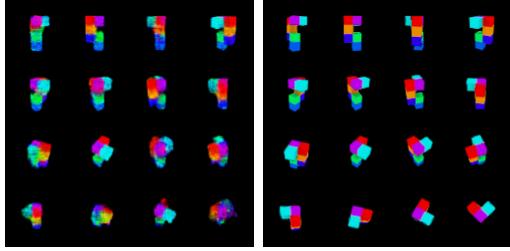


Figure 2: Shepard-Metzler object from 1k-object training set, 15 observations each. SRNs (right) outperform dGQN (left) on this small dataset.

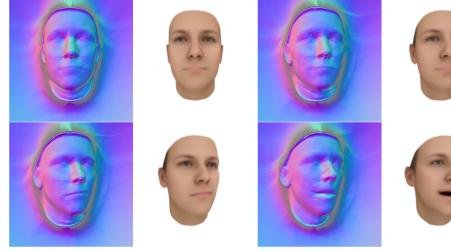


Figure 3: Non-rigid animation of a face. Note that mouth movement is directly reflected in the normal maps.

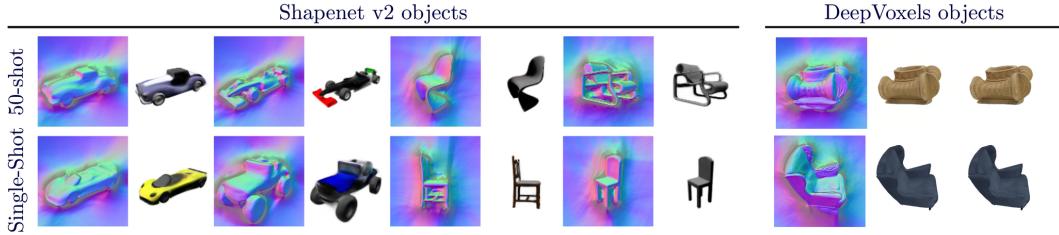


Figure 4: Normal maps for a selection of objects. We note that geometry is learned fully unsupervised and arises purely out of the perspective and multi-view geometry constraints on the image formation.

### 3.4 Joint Optimization

To summarize, given a dataset  $\mathcal{D} = \{\mathcal{C}_j\}_{j=1}^M$  of instance datasets  $\mathcal{C} = \{(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i)\}_{i=1}^N$ , we aim to find the parameters  $\psi$  of  $\Psi$  that maps latent vectors  $\mathbf{z}_j$  to the parameters of the respective scene representation  $\phi_j$ , the parameters  $\theta$  of the neural rendering function  $\Theta$ , as well as the latent codes  $\mathbf{z}_j$  themselves. We formulate this as an optimization problem with the following objective:

$$\arg \min_{\{\theta, \psi, \{\mathbf{z}_j\}_{j=1}^M\}} \sum_{j=1}^M \sum_{i=1}^N \underbrace{\|\Theta_\theta(\Phi_{\Psi(\mathbf{z}_j)}, \mathbf{E}_i^j, \mathbf{K}_i^j) - \mathcal{I}_i^j\|_2^2}_{\mathcal{L}_{\text{img}}} + \underbrace{\lambda_{\text{dep}} \|\min(\mathbf{d}_{i,f\text{inal}}^j, \mathbf{0})\|_2^2}_{\mathcal{L}_{\text{depth}}} + \underbrace{\lambda_{\text{lat}} \|\mathbf{z}_j\|_2^2}_{\mathcal{L}_{\text{latent}}}. \quad (6)$$

Where  $\mathcal{L}_{\text{img}}$  is an  $\ell_2$ -loss enforcing closeness of the rendered image to ground-truth,  $\mathcal{L}_{\text{depth}}$  is a regularization term that accounts for the positivity constraint in Eq. 4, and  $\mathcal{L}_{\text{latent}}$  enforces a Gaussian prior on the  $\mathbf{z}_j$ . In the case of a single scene, this objective simplifies to solving for the parameters  $\phi$  of the MLP parameterization of  $\Phi$  instead of the parameters  $\psi$  and latent codes  $\mathbf{z}_j$ . We solve Eq. 6 with stochastic gradient descent. Note that the whole pipeline can be trained end-to-end, without requiring any (pre-)training of individual parts. In Sec. 4, we demonstrate that SRNs discover both geometry and appearance, initialized at random, without requiring prior knowledge of either scene geometry or scene scale, enabling multi-view consistent novel view synthesis.

**Few-shot reconstruction.** After finding model parameters by solving Eq. 6, we may use the trained model for few-shot reconstruction of a new object instance, represented by a dataset  $\mathcal{C} = \{(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i)\}_{i=1}^N$ . We fix  $\theta$  as well as  $\psi$ , and estimate a new latent code  $\hat{\mathbf{z}}$  by minimizing

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{z}} \sum_{i=1}^N \|\Theta_\theta(\Phi_{\Psi(\mathbf{z})}, \mathbf{E}_i, \mathbf{K}_i) - \mathcal{I}_i\|_2^2 + \lambda_{\text{dep}} \|\min(\mathbf{d}_{i,f\text{inal}}, \mathbf{0})\|_2^2 + \lambda_{\text{lat}} \|\mathbf{z}\|_2^2 \quad (7)$$

## 4 Experiments

We train SRNs on several object classes and evaluate them for novel view synthesis and few-shot reconstruction. We further demonstrate the discovery of a non-rigid face model. Please see the supplement for a comparison on single-scene novel view synthesis performance with DeepVoxels [6].



Figure 5: Interpolating latent code vectors of cars and chairs in the Shapenet dataset while rotating the camera around the model. Features smoothly transition from one model to another.



Figure 6: Qualitative comparison with Tatarchenko et al. [1] and the deterministic variant of the GQN [2], for novel view synthesis on the Shapenet v2 “cars” and “chairs” classes. We compare novel views for objects reconstructed from 50 observations in the training set (top row), two observations and a single observation (second and third row) from a test set. SRNs consistently outperforms these baselines with multi-view consistent novel views, while also reconstructing geometry. Please see the supplemental video for more comparisons, smooth camera trajectories, and reconstructed geometry.

**Implementation Details.** Hyperparameters, computational complexity, and full network architectures for SRNs and all baselines are in the supplement. Training of the presented models takes on the order of 6 days. A single forward pass takes around 120 ms and 3 GB of GPU memory per batch item. Code and datasets are available.

**Shepard-Metzler objects.** We evaluate our approach on 7-element Shepard-Metzler objects in a limited-data setting. We render 15 observations of 1k objects at a resolution of  $64 \times 64$ . We train both SRNs and a deterministic variant of the Generative Query Network [2] (dGQN, please see supplement for an extended discussion). Note that the dGQN is solving a harder problem, as it is inferring the scene representation in each forward pass, while our formulation requires solving an optimization problem to find latent codes for unseen objects. We benchmark novel view reconstruction accuracy on (1) the training set and (2) few-shot reconstruction of 100 objects from a held-out test set. On the training objects, SRNs achieve almost pixel-perfect results with a PSNR of 30.41 dB. The dGQN fails to learn object shape and multi-view geometry on this limited dataset, achieving 20.85 dB. See Fig. 2 for a qualitative comparison. In a two-shot setting (see Fig. 7 for reference views), we succeed in reconstructing any part of the object that has been observed, achieving 24.36 dB, while the dGQN achieves 18.56 dB. In a one-shot setting, SRNs reconstruct an object consistent with the observed view. As expected, due to the current non-probabilistic implementation, both the dGQN and SRNs reconstruct an object resembling the mean of the hundreds of feasible objects that may have generated the observation, achieving 17.51 dB and 18.11 dB respectively.

**Shapenet v2.** We consider the “chair” and “car” classes of Shapenet v.2 [39] with 4.5k and 2.5k model instances respectively. We disable transparencies and specularities, and train on 50 observations of each instance at a resolution of  $128 \times 128$  pixels. Camera poses are randomly generated on a sphere with the object at the origin. We evaluate performance on (1) novel-view synthesis of objects in the training set and (2) novel-view synthesis on objects in the held-out, official Shapenet v2 test sets, reconstructed from one or two observations, as discussed in Sec. 3.4. Fig. 7 shows the sampled poses for the few-shot case. In all settings, we assemble ground-truth novel views by sampling 250 views in an Archimedean spiral around each object instance. We compare



Figure 7: Single- (left) and two-shot (both) reference views.

Table 1: PSNR (in dB) and SSIM of images reconstructed with our method, the deterministic variant of the GQN [2] (dGQN), the model proposed by Tatarchenko et al. [1] (TCO), and the method proposed by Worrall et al. [4] (WRL). We compare novel-view synthesis performance on objects in the training set (containing 50 images of each object), as well as reconstruction from 1 or 2 images on the held-out test set.

	50 images (training set)		2 images		Single image	
	Chairs	Cars	Chairs	Cars	Chairs	Cars
TCO [1]	24.31 / 0.92	20.38 / 0.83	21.33 / 0.88	18.41 / 0.80	21.27 / 0.88	18.15 / 0.79
WRL [4]	24.57 / 0.93	19.16 / 0.82	22.28 / 0.90	17.20 / 0.78	22.11 / 0.90	16.89 / 0.77
dGQN [2]	22.72 / 0.90	19.61 / 0.81	22.36 / 0.89	18.79 / 0.79	21.59 / 0.87	18.19 / 0.78
SRNs	<b>26.23 / 0.95</b>	<b>26.32 / 0.94</b>	<b>24.48 / 0.92</b>	<b>22.94 / 0.88</b>	<b>22.89 / 0.91</b>	<b>20.72 / 0.85</b>

SRNs to three baselines from recent literature. Table 1 and Fig. 6 report quantitative and qualitative results respectively. In all settings, we outperform all baselines by a wide margin. On the training set, we achieve very high visual fidelity. Generally, views are perfectly multi-view consistent, the only exception being objects with distinct, usually fine geometric detail, such as the windscreen of convertibles. None of the baselines succeed in generating multi-view consistent views. Several views per object are usually entirely degenerate. In the two-shot case, where most of the object has been seen, SRNs still reconstruct both object appearance and geometry robustly. In the single-shot case, SRNs complete unseen parts of the object in a plausible manner, demonstrating that the learned priors have truthfully captured the underlying distributions.

**Supervising parameters for non-rigid deformation.** If latent parameters of the scene are known, we can condition on these parameters instead of jointly solving for latent variables  $\mathbf{z}_j$ . We generate 50 renderings each from 1000 faces sampled at random from the Basel face model [71]. Camera poses are sampled from a hemisphere in front of the face. Each face is fully defined by a 224-dimensional parameter vector, where the first 160 parameterize identity, and the last 64 dimensions control facial expression. We use a constant ambient illumination to render all faces. Conditioned on this disentangled latent space, SRNs succeed in reconstructing face geometry and appearance. After training, we animate facial expression by varying the 64 expression parameters while keeping the identity fixed, even though this specific combination of identity and expression has not been observed before. Fig. 3 shows qualitative results of this non-rigid deformation. Expressions smoothly transition from one to the other, and the reconstructed normal maps, which are directly computed from the depth maps (not shown), demonstrate that the model has learned the underlying geometry.

**Geometry reconstruction.** SRNs reconstruct geometry in a fully unsupervised manner, purely out of necessity to explain observations in 3D. Fig. 4 visualizes geometry for 50-shot, single-shot, and single-scene reconstructions.

**Latent space interpolation.** Our learned latent space allows meaningful interpolation of object instances. Fig. 5 shows latent space interpolation.

**Pose extrapolation.** Due to the explicit 3D-aware and per-pixel formulation, SRNs naturally generalize to 3D transformations that have never been seen during training, such as camera close-ups or camera roll, even when trained only on up-right camera poses distributed on a sphere around the objects. Please see the supplemental video for examples of pose extrapolation.

**Failure cases.** The ray marcher may “get stuck” in holes of surfaces or on rays that closely pass by occluders, such as commonly occur in chairs. SRNs generates a continuous surface in these cases, or will sometimes step through the surface. If objects are far away from the training distribution, SRNs may fail to reconstruct geometry and instead only match texture. In both cases, the reconstructed geometry allows us to analyze the failure, which is impossible with black-box alternatives. See Fig. 8 and the supplemental video.

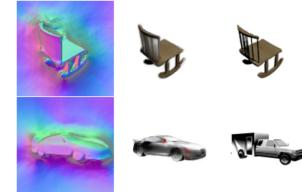


Figure 8: Failure cases.

**Towards representing room-scale scenes.** We demonstrate reconstruction of a room-scale scene with SRNs. We train a single SRN on 500 observations of a minecraft room. The room contains multiple objects as well as four columns, such that parts of the scene are occluded in most observations. After training, the SRN enables novel view synthesis of the room. Though generated images are blurry, they are largely multi-view consistent, with artifacts due to ray marching failures only at object boundaries and thin structures. The SRN succeeds in inferring geometry and appearance of the room, reconstructing occluding columns and objects correctly, failing only on low-texture areas (where geometry is only weakly constrained) and thin tubes placed between columns. Please see the supplemental video for qualitative results.

## 5 Discussion

We introduce SRNs, a 3D-structured neural scene representation that implicitly represents a scene as a continuous, differentiable function. This function maps 3D coordinates to a feature-based representation of the scene and can be trained end-to-end with a differentiable ray marcher to render the feature-based representation into a set of 2D images. SRNs do not require shape supervision and can be trained only with a set of posed 2D images. We demonstrate results for novel view synthesis, shape and appearance interpolation, and few-shot reconstruction.

There are several exciting avenues for future work. SRNs could be explored in a probabilistic framework [2, 3], enabling sampling of feasible scenes given a set of observations. SRNs could be extended to model view- and lighting-dependent effects, translucency, and participating media. They could also be extended to other image formation models, such as computed tomography or magnetic resonance imaging. Currently, SRNs require camera intrinsic and extrinsic parameters, which can be obtained robustly via bundle-adjustment. However, as SRNs are differentiable with respect to camera parameters; future work may alternatively integrate them with learned algorithms for camera pose estimation [72]. SRNs also have exciting applications outside of vision and graphics, and future work may explore SRNs in robotic manipulation or as the world model of an independent agent. While SRNs can represent room-scale scenes (see the supplemental video), generalization across complex, cluttered 3D environments is an open problem. Recent work in meta-learning could enable generalization across scenes with weaker assumptions on the dimensionality of the underlying manifold [73]. Please see the supplemental material for further details on directions for future work.

## 6 Acknowledgements

We thank Ludwig Schubert and Oliver Groth for fruitful discussions. Vincent Sitzmann was supported by a Stanford Graduate Fellowship. Michael Zollhöfer was supported by the Max Planck Center for Visual Computing and Communication (MPC-VCC). Gordon Wetzstein was supported by NSF awards (IIS 1553333, CMMI 1839974), by a Sloan Fellowship, by an Okawa Research Grant, and a PECASE.

## References

- [1] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Single-view to multi-view: Reconstructing unseen views with a convolutional network,” *CoRR abs/1511.06702*, vol. 1, no. 2, p. 2, 2015.
- [2] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor *et al.*, “Neural scene representation and rendering,” *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.
- [3] A. Kumar, S. A. Eslami, D. Rezende, M. Garnelo, F. Viola, E. Lockhart, and M. Shanahan, “Consistent jumpy predictions for videos and scenes,” 2018.
- [4] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, “Interpretable transformations with encoder-decoder networks,” in *Proc. ICCV*, vol. 4, 2017.
- [5] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *Proc. IROS*, September 2015, p. 922 – 928.
- [6] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “Deepvoxels: Learning persistent 3d feature embeddings,” in *Proc. CVPR*, 2019.

- [7] A. Kar, C. Häne, and J. Malik, “Learning a multi-view stereo machine,” in *Proc. NIPS*, 2017, pp. 365–376.
- [8] H.-Y. F. Tung, R. Cheng, and K. Fragkiadaki, “Learning spatial common sense with geometry-aware recurrent networks,” *Proc. CVPR*, 2019.
- [9] T. H. Nguyen-Phuoc, C. Li, S. Balaban, and Y. Yang, “Rendernet: A deep convolutional network for differentiable rendering from 3d shapes,” in *Proc. NIPS*, 2018.
- [10] J.-Y. Zhu, Z. Zhang, C. Zhang, J. Wu, A. Torralba, J. Tenenbaum, and B. Freeman, “Visual object networks: image generation with disentangled 3d representations,” in *Proc. NIPS*, 2018, pp. 118–129.
- [11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” *Proc. CVPR*, 2017.
- [12] E. Insafutdinov and A. Dosovitskiy, “Unsupervised learning of shape and pose with differentiable point clouds,” in *Proc. NIPS*, 2018, pp. 2802–2812.
- [13] M. Meshry, D. B. Goldman, S. Khamis, H. Hoppe, R. Pandey, N. Snavely, and R. Martin-Brualla, “Neural rerendering in the wild,” *Proc. CVPR*, 2019.
- [14] C.-H. Lin, C. Kong, and S. Lucey, “Learning efficient point cloud generation for dense 3d object reconstruction,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [15] D. Jack, J. K. Pontes, S. Sridharan, C. Fookes, S. Shirazi, F. Maire, and A. Eriksson, “Learning free-form deformations for 3d object reconstruction,” *CoRR*, 2018.
- [16] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, “Multi-view supervision for single-view reconstruction via differentiable ray consistency,” in *Proc. CVPR*.
- [17] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling,” in *Proc. NIPS*, 2016, pp. 82–90.
- [18] M. Gadelha, S. Maji, and R. Wang, “3d shape induction from 2d views of multiple objects,” in *3DV*. IEEE Computer Society, 2017, pp. 402–411.
- [19] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas, “Volumetric and multi-view cnns for object classification on 3d data,” in *Proc. CVPR*, 2016.
- [20] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman, “Pix3d: Dataset and methods for single-image 3d shape modeling,” in *Proc. CVPR*, 2018.
- [21] D. Jimenez Rezende, S. M. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, “Unsupervised learning of 3d structure from images,” in *Proc. NIPS*, 2016.
- [22] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3d-r2n2: A unified approach for single and multi-view 3d object reconstruction,” in *Proc. ECCV*, 2016.
- [23] G. Riegler, A. O. Ulusoy, and A. Geiger, “Octnet: Learning deep 3d representations at high resolutions,” in *Proc. CVPR*, 2017.
- [24] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs,” in *Proc. ICCV*, 2017, pp. 2107–2115.
- [25] C. Haene, S. Tulsiani, and J. Malik, “Hierarchical surface prediction,” *Proc. PAMI*, pp. 1–1, 2019.
- [26] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, “Learning representations and generative models for 3D point clouds,” in *Proc. ICML*, 2018, pp. 40–49.
- [27] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Multi-view 3d models from single images with a convolutional network,” in *Proc. ECCV*, 2016.
- [28] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, “Stereo magnification: learning view synthesis using multiplane images,” *ACM Trans. Graph.*, vol. 37, no. 4, pp. 65:1–65:12, 2018.
- [29] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, “Atlasnet: A papier-mâché approach to learning 3d surface generation,” in *Proc. CVPR*, 2018.
- [30] H. Kato, Y. Ushiku, and T. Harada, “Neural 3d mesh renderer,” in *Proc. CVPR*, 2018, pp. 3907–3916.
- [31] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik, “Learning category-specific mesh reconstruction from image collections,” in *ECCV*, 2018.

- [32] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3d reconstruction in function space,” in *Proc. CVPR*, 2019.
- [33] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, “Deepsdf: Learning continuous signed distance functions for shape representation,” *arXiv preprint arXiv:1901.05103*, 2019.
- [34] K. Genova, F. Cole, D. Vlasic, A. Sarna, W. T. Freeman, and T. Funkhouser, “Learning shape templates with structured implicit functions,” *Proc. ICCV*, 2019.
- [35] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi, “Cvxnets: Learnable convex decomposition,” *arXiv preprint arXiv:1909.05736*, 2019.
- [36] T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y. Yang, “Hologan: Unsupervised learning of 3d representations from natural images,” in *Proc. ICCV*, 2019.
- [37] F. Alet, A. K. Jeewajee, M. Bauza, A. Rodriguez, T. Lozano-Perez, and L. P. Kaelbling, “Graph element networks: adaptive, structured computation and memory,” in *Proc. ICML*, 2019.
- [38] Y. Liu, Z. Wu, D. Ritchie, W. T. Freeman, J. B. Tenenbaum, and J. Wu, “Learning to describe scenes with programs,” in *Proc. ICLR*, 2019.
- [39] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, “Shapenet: An information-rich 3d model repository,” *arXiv preprint arXiv:1512.03012*, 2015.
- [40] G. E. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.
- [41] D. P. Kingma and M. Welling, “Auto-encoding variational bayes.” in *Proc. ICLR*, 2013.
- [42] L. Dinh, D. Krueger, and Y. Bengio, “NICE: non-linear independent components estimation,” in *Proc. ICLR Workshops*, 2015.
- [43] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” in *NeurIPS*, 2018, pp. 10 236–10 245.
- [44] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, “Conditional image generation with pixelcnn decoders,” in *Proc. NIPS*, 2016, pp. 4797–4805.
- [45] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *Proc. ICML*, 2016.
- [46] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Proc. NIPS*, 2014.
- [47] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proc. ICML*, 2017.
- [48] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” in *Proc. ICLR*, 2018.
- [49] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, “Generative visual manipulation on the natural image manifold,” in *Proc. ECCV*, 2016.
- [50] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *Proc. ICLR*, 2016.
- [51] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” 2014, arXiv:1411.1784.
- [52] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proc. CVPR*, 2017, pp. 5967–5976.
- [53] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proc. ICCV*, 2017.
- [54] K. O. Stanley, “Compositional pattern producing networks: A novel abstraction of development,” *Genetic programming and evolvable machines*, vol. 8, no. 2, pp. 131–162, 2007.
- [55] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah, “Differentiable image parameterizations,” *Distill*, vol. 3, no. 7, p. e12, 2018.

- [56] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, “Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision,” in *Proc. NIPS*, 2016.
- [57] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, “Spatial transformer networks,” in *Proc. NIPS*, 2015.
- [58] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming auto-encoders,” in *Proc. ICANN*, 2011.
- [59] A. Yuille and D. Kersten, “Vision as Bayesian inference: analysis by synthesis?” *Trends in Cognitive Sciences*, vol. 10, pp. 301–308, 2006.
- [60] T. Bever and D. Poeppel, “Analysis by synthesis: A (re-)emerging program of research for language and vision,” *Biolinguistics*, vol. 4, no. 2, pp. 174–200, 2010.
- [61] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, “Deep convolutional inverse graphics network,” in *Proc. NIPS*, 2015.
- [62] J. Yang, S. Reed, M.-H. Yang, and H. Lee, “Weakly-supervised disentangling with recurrent transformations for 3d view synthesis,” in *Proc. NIPS*, 2015.
- [63] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka, “Picture: A probabilistic programming language for scene perception,” in *Proc. CVPR*, 2015.
- [64] H. F. Tung, A. W. Harley, W. Seto, and K. Fragniadaki, “Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision,” in *Proc. ICCV*.
- [65] Z. Shu, E. Yumer, S. Hadap, K. Sunkavalli, E. Shechtman, and D. Samaras, “Neural face editing with intrinsic image disentangling,” in *Proc. CVPR*, 2017.
- [66] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003.
- [67] J. C. Hart, “Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces,” *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.
- [68] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves *et al.*, “Conditional image generation with pixelcnn decoders,” in *Proc. NIPS*, 2016.
- [69] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [70] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” in *Proc. ICLR*, 2017.
- [71] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter, “A 3d face model for pose and illumination invariant face recognition,” in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*. Ieee, 2009, pp. 296–301.
- [72] C. Tang and P. Tan, “Ba-net: Dense bundle adjustment network,” in *Proc. ICLR*, 2019.
- [73] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proc. ICML*. JMLR.org, 2017, pp. 1126–1135.

---

# Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations

## -Supplementary Material-

---

Vincent Sitzmann   Michael Zollhöfer   Gordon Wetzstein  
{sitzmann, zollhoefer}@cs.stanford.edu, gordon.wetzstein@stanford.edu  
Stanford University

## Contents

<b>1 Additional Results on Neural Ray Marching</b>	<b>2</b>
<b>2 Comparison to DeepVoxels</b>	<b>2</b>
<b>3 Reproducibility</b>	<b>4</b>
3.1 Architecture Details . . . . .	4
3.2 Time & Memory Complexity . . . . .	5
3.3 Dataset Details . . . . .	5
3.4 SRNs Training Details . . . . .	6
3.4.1 General details . . . . .	6
3.4.2 Per-experiment details . . . . .	6
<b>4 Relationship to per-pixel autoregressive methods</b>	<b>6</b>
<b>5 Baseline Discussions</b>	<b>7</b>
5.1 Deterministic Variant of GQN . . . . .	7
5.2 Tatarchenko et al. . . . .	8
5.3 Worrall et al. . . . .	8
<b>6 Differentiable Ray-Marching in the context of classical renderers</b>	<b>9</b>
<b>7 Trade-offs of the Pixel Generator vs. CNN-based renderers</b>	<b>9</b>
<b>8 Future work</b>	<b>9</b>

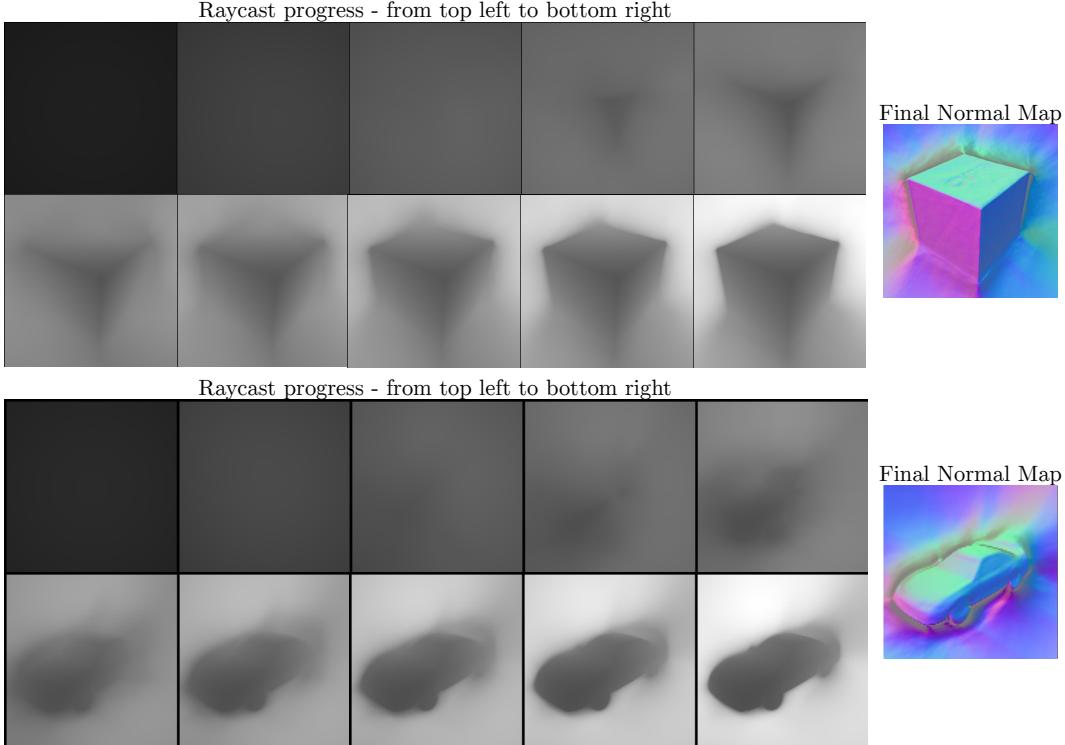


Figure 1: Visualizations of ray marching progress and the final normal map. Note that the uniformly colored background does not constrain the depth - as a result, the depth is unconstrained around the silhouette of the object. Since the final normal map visualizes surface detail much better, we only report the final normal map in the main document.

## 1 Additional Results on Neural Ray Marching

**Computation of Normal Maps** We found that normal maps visualize fine surface detail significantly better than depth maps (see Fig. 1), and thus only report normal maps in the main submission. We compute surface normals as the cross product of the numerical horizontal and vertical derivatives of the depth map.

**Ray Marching Progress Visualization** The  $z$ -coordinates of running and final estimates of intersections in each iteration of the ray marcher in camera coordinates yield depth maps, which visualize every step of the ray marcher. Fig. 1 shows two example ray marches, along with their final normal maps.

## 2 Comparison to DeepVoxels

We compare performance in single-scene novel-view synthesis with the recently proposed DeepVoxels architecture [1] on their four synthetic objects. DeepVoxels proposes a 3D-structured neural scene representation in the form of a voxel grid of features. Multi-view and projective geometry are hard-coded into the model architecture. We further report accuracy of the same baselines as in [1]: a Pix2Pix architecture [2] that receives as input the per-pixel view direction, as well as the methods proposed by Tatarchenko et al. [3] as well as by Worrall et al. [4] and Cohen and Welling [5].

Table 1 compares PSNR and SSIM of the proposed architecture and the baselines, averaged over all 4 scenes. We outperform the best baseline, DeepVoxels [1], by more than 3 dB. Qualitatively, DeepVoxels displays significant multi-view inconsistencies in the form of flickering artifacts, while the proposed method is almost perfectly multi-view consistent. We achieve this result with 550k parameters per model, as opposed to the DeepVoxels architecture with more than 160M free variables. However, we found that SRNs produce blurry output for some of the very high-frequency textural

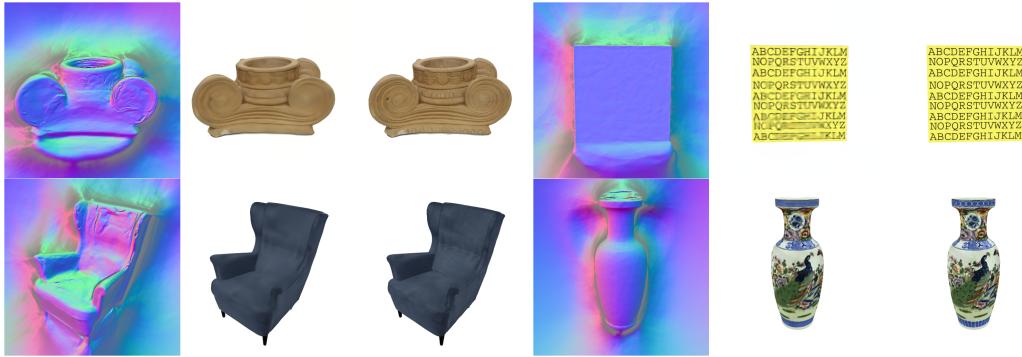


Figure 2: Qualitative results on DeepVoxels objects. For each object: Left: Normal map of reconstructed geometry. Center: SRNs output. Right: Ground Truth.

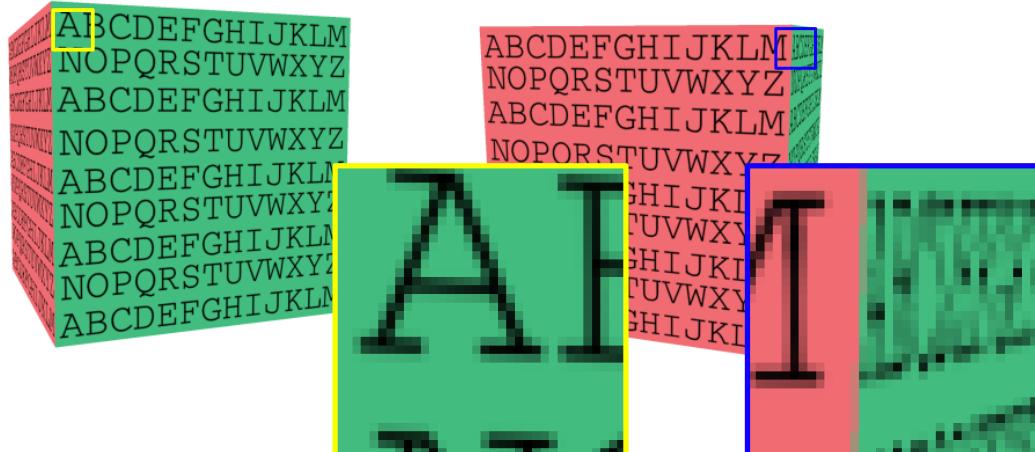


Figure 3: Undersampled letters on the side of the cube (ground truth images). Lines of letters are less than two pixels wide, leading to significant aliasing. Additionally, the 2D downsampling as described in [1] introduced blur that is not multi-view consistent.



Figure 4: By using a U-Net renderer similar to [1], we can reconstruct the undersampled letters. In exchange, we lose the guarantee of multi-view consistency. Left: Reconstructed normal map. Center: SRNs output. Right: ground truth.

	PSNR	SSIM
Tatarchenko et al. [3]	21.22	0.90
Worrall et al. [4]	21.22	0.90
Pix2Pix [2]	23.63	0.92
DeepVoxels [1]	30.55	0.97
SRNs	<b>33.03</b>	<b>0.97</b>

Table 1: Quantitative comparison to DeepVoxels [1]. With 3 orders of magnitude fewer parameters, we achieve a 3dB boost, with reduced multi-view inconsistencies.

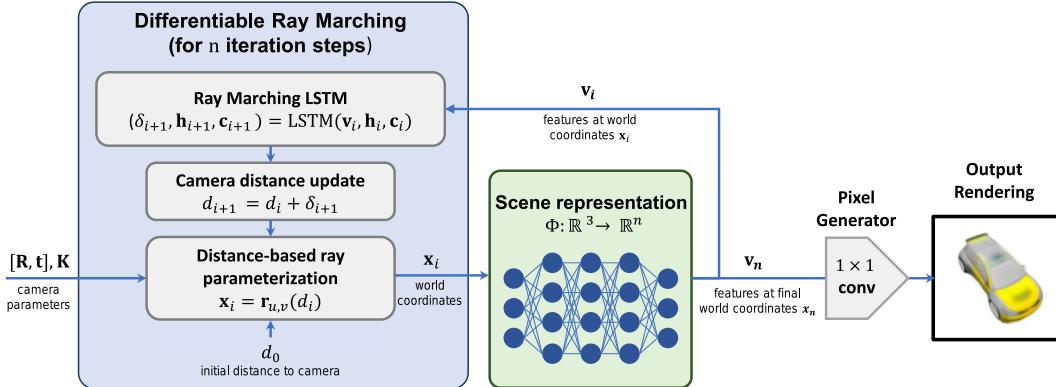


Figure 5: Architecture overview: at the heart of SRNs lies a continuous, 3D-aware neural scene representation,  $\Phi$ , which represents a scene as a function that maps  $(x, y, z)$  world coordinates to a feature representation of the scene at those coordinates. To render  $\Phi$ , a neural ray-marcher interacts with  $\Phi$  via world coordinates along camera rays, parameterized via their distance  $d$  to the camera projective center. Ray Marching begins at a distance  $d_0$  close to the camera. In each step, the scene representation network  $\Phi$  is queried at the current world coordinates  $x_i$ . The resulting feature vector  $v_i$  is fed to the Ray Marching LSTM that predicts a step length  $\delta_{i+1}$ . The world coordinates are updated according to the new distance to the camera,  $d_{i+1} = d_i + \delta_{i+1}$ . This is repeated for a fixed number of iterations,  $n$ . The features at the final world coordinates  $v_n = \Phi(x_n)$  are then translated to an RGB color by the pixel generator.

detail - this is most notable with the letters on the sides of the cube. Fig. 3 demonstrates why this is the case. Several of the high-frequency textural detail of the DeepVoxels objects are heavily undersampled. For instance, lines of letters on the sides of the cube often only occupy a single pixel. As a result, the letters alias across viewing angles. This violates one of our key assumptions, namely that the same  $(x, y, z) \in \mathbb{R}^3$  world coordinate always maps to the same color, independent of the viewing angle. As a result, it is impossible for our model to generate these details. We note that detail that is not undersampled, such as the CVPR logo on the top of the cube, is reproduced with perfect accuracy. However, we can easily accommodate for this undersampling by using a 2D CNN renderer. This amounts to a trade-off of our guarantee of multi-view consistency discussed in Sec. 3 of the main paper with robustness to faulty training data. Fig. 2 shows the cube rendered with a U-Net based renderer – all detail is replicated truthfully.

### 3 Reproducibility

In this section, we discuss steps we take to allow the community to reproduce our results. *All code and datasets will be made publicly available.* All models were evaluated on the test sets exactly once.

#### 3.1 Architecture Details

**Scene representation network  $\Phi$**  In all experiments,  $\Phi$  is parameterized as a multi-layer perceptron (MLP) with ReLU activations, layer normalization before each nonlinearity [6], and four layers with 256 units each. In all generalization experiments in the main paper, its weights  $\phi$  are the output of the

hypernetwork  $\Psi$ . In the DeepVoxels comparison (see Sec.2), where a separate  $\Phi$  is trained per scene, parameters of  $\phi$  are directly initialized using the Kaiming Normal method [7].

**Hypernetwork  $\Psi$**  In generalization experiments, a hypernetwork  $\Psi$  maps a latent vector  $\mathbf{z}_j$  to the weights of the respective scene representation  $\phi_j$ . Each layer of  $\Phi$  is the output of a separate hypernetwork. Each hypernetwork is parameterized as a multi-layer perceptron with ReLU activations, layer normalization before each nonlinearity [6], and three layers (where the last layer has as many units as the respective layer of  $\Phi$  has weights). In the Shapenet and Shepard-Metzler experiments, where the latent codes  $\mathbf{z}_j$  have length 256, hypernetworks have 256 units per layer. In the Basel face experiment, where the latent codes  $\mathbf{z}_j$  have length 224, hypernetworks have 224 units per layer. Weights are initialized by the Kaiming Normal method, scaled by a factor 0.1. We empirically found this initialization to stabilize early training.

**Ray marching LSTM** In all experiments, the ray marching LSTM is implemented as a vanilla LSTM with a hidden state size of 16. The initial state is set to zero.

**Pixel Generator** In all experiments, the pixel generator is parameterized as a multi-layer perceptron with ReLU activations, layer normalization before each nonlinearity [6], and five layers with 256 units each. Weights are initialized with the Kaiming Normal method [7].

### 3.2 Time & Memory Complexity

**Scene representation network  $\Phi$**   $\Phi$  scales as a standard MLP. Memory and runtime scale linearly in the number of queries, therefore quadratic in image resolution. Memory and runtime further scale linearly with the number of layers and quadratically with the number of units in each layer.

**Hypernetwork  $\Psi$**   $\Psi$  scales as a standard MLP. Notably, the last layer of  $\Psi$  predicts all parameters of the scene representation  $\Phi$ . As a result, the number of weights scales linearly in the number of weights of  $\Phi$ , which is significant. For instance, with 256 units per layer and 4 layers,  $\Phi$  has approximately  $2 \times 10^5$  parameters. In our experiments,  $\Psi$  is parameterized with 256 units in all hidden layers. The last layer of  $\Psi$  then has approximately  $5 \times 10^7$  parameters, which is the bulk of learnable parameters in our model. Please note that  $\Psi$  only has to be queried once to obtain  $\Phi$ , at which point it could be discarded, as both the pixel generation and the ray marching only need access to the predicted  $\Phi$ .

**Differentiable Ray Marching** Memory and runtime of the differentiable ray marcher scale linearly in the number of ray marching steps and quadratically in image resolution. As it queries  $\Phi$  repeatedly, it also scales linearly in the same parameters as  $\Phi$ .

**Pixel Generator** The pixel generator scales as a standard MLP. Memory and runtime scale linearly in the number of queries, therefore quadratic in image resolution. Memory and runtime further scale linearly with the number of layers and quadratically with the number of units in each layer.

### 3.3 Dataset Details

**Shepard-Metzler objects** We modified an open-source implementation of a Shepard-Metzler renderer (<https://github.com/musyoku/gqn-dataset-renderer.git>) to generate meshes of Shepard-Metzler objects, which we rendered using Blender to have full control over camera intrinsic and extrinsic parameters consistent with other presented datasets.

**Shapenet v2 cars** We render each object from random camera perspectives distributed on a sphere with radius 1.3 using Blender. We disabled specularities, shadows and transparencies and used environment lighting with energy 1.0. We noticed that a few cars in the dataset were not scaled optimally, and scaled their bounding box to unit length. A few meshes had faulty vertices, resulting in a faulty bounding box and subsequent scaling to a very small size. We discarded those 40 out of 2473 cars.

**Shapenet v2 chairs** We render each object from random camera perspectives distributed on a sphere with radius 2.0 using Blender. We disabled specularities, shadows and transparencies and used environment lighting with energy 1.0.

**Faces dataset** We use the Basel Face dataset to generate meshes with different identities at random, where each parameter is sampled from a normal distribution with mean 0 and standard deviation of 0.7. For expressions, we use the blendshape model of Thies et al. [8], and sample expression parameters uniformly in  $(-0.4, 1.6)$ .

**DeepVoxels dataset** We use the dataset as presented in [1].

### 3.4 SRNs Training Details

#### 3.4.1 General details

**Multi-Scale training** Our per-pixel formulation naturally allows us to train in a coarse-to-fine setting, where we first train the model on downsampled images in a first stage, and then increase the resolution of images in stages. This allows larger batch sizes at the beginning of the training, which affords more independent views for each object, and is reminiscent of other coarse-to-fine approaches [9].

**Solver** For all experiments, we use the ADAM solver with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ .

**Implementation & Compute** We implement all models in PyTorch. All models were trained on single GPUs of the type RTX6000 or RTX8000.

**Hyperparameter search** Training hyperparameters for SRNs were found by informal search – we did not perform a systematic grid search due to the high computational cost.

#### 3.4.2 Per-experiment details

For a resolution of  $64 \times 64$ , we train with a batch size of 72. Due to the memory complexity being quadratic in the image sidelength, we decrease the batch size by a factor of 4 when we double the image resolution.  $\lambda_{\text{depth}}$  is always set to  $1 \times 10^{-3}$  and  $\lambda_{\text{latent}}$  is set to 1. The ADAM learning rate is set to  $4 \times 10^{-4}$  if not reported otherwise.

**Shepard-Metzler experiment** We directly train our model on images of resolution  $64 \times 64$  for 352 epochs.

**Shapenet cars** We train our model in 2 stages. We first train on a resolution of  $64 \times 64$  for 5k iterations. We then increase the resolution to  $128 \times 128$ . We train on the high resolution for 70 epochs. The ADAM learning rate is set to  $5 \times 10^{-5}$ .

**Shapenet chairs** We train our model in 2 stages. We first train on a resolution of  $64 \times 64$  for 20k iterations. We then increase the resolution to  $128 \times 128$ . We train our model for 12 epochs.

**Basel face experiments** We train our model in 2 stages. We first train on a resolution of  $64 \times 64$  for 15k iterations. We then increase the resolution to  $128 \times 128$  and train for another 5k iterations.

**DeepVoxels experiments** We train our model in 3 stages. We first train on a resolution of  $12 \times 128$  with a learning rate of  $4 \times 10^{-4}$  for 20k iterations. We then increase the resolution to  $256 \times 256$ , and lower the learning rate to  $1 \times 10^{-4}$  and train for another 30k iterations. We then increase the resolution to  $512 \times 512$ , and lower the learning rate to  $4 \times 10^{-6}$  and train for another 30k iterations.

## 4 Relationship to per-pixel autoregressive methods

With the proposed per-pixel generator, SRNs are also reminiscent of autoregressive per-pixel architectures, such as PixelCNN and PixelRNN [10, 11]. The key difference to autoregressive per-pixel

architectures lies in the modeling of the probability  $p(\mathcal{I})$  of an image  $\mathcal{I} \in \mathbb{R}^{H \times W \times 3}$ . PixelCNN and PixelRNN model an image as a one-dimensional sequence of pixel values  $\mathcal{I}_1, \dots, \mathcal{I}_{H \times W}$ , and estimate their joint distribution as

$$p(\mathcal{I}) = \prod_{i=1}^{H \times W} p(\mathcal{I}_i | \mathcal{I}_1, \dots, \mathcal{I}_{i-1}). \quad (1)$$

Instead, conditioned on a scene representation  $\Phi$ , pixel values are conditionally independent, as our approach independently and deterministically assigns a value to each pixel. The probability of observing an image  $\mathcal{I}$  thus simplifies to the probability of observing a scene  $\Phi$  under extrinsic  $\mathbf{E}$  and intrinsic  $\mathbf{K}$  camera parameters

$$p(\mathcal{I}) = p(\Phi)p(\mathbf{E})p(\mathbf{K}). \quad (2)$$

This conditional independence of single pixels conditioned on the scene representation further motivates the per-pixel design of the rendering function  $\Theta$ .

## 5 Baseline Discussions

### 5.1 Deterministic Variant of GQN

**Deterministic vs. Non-Deterministic** Eslami et al. [12] propose a powerful probabilistic framework for modeling uncertainty in the reconstruction due to incomplete observations. However, here, we are exclusively interested in investigating the properties of the scene representation itself, and this submission discusses SRNs in a purely deterministic framework. To enable a fair comparison, we thus implement a deterministic baseline inspired by the Generative Query Network [12]. We note that the results obtained in this comparison are not necessarily representative of the performance of the unaltered Generative Query Network. We leave a formulation of SRNs in a probabilistic framework and a comparison to the unaltered GQN to future work.

**Architecture** As representation network architecture, we choose the "Tower" representation, and leave its architecture unaltered. However, instead of feeding the resulting scene representation  $\mathbf{r}$  to a convolutional LSTM architecture to parameterize a density over latent variables  $\mathbf{z}$ , we instead directly feed the scene representation  $\mathbf{r}$  to a generator network. We use as generator a deterministic, autoregressive, skip-convolutional LSTM  $C$ , the deterministic equivalent of the generator architecture proposed in [12]. Specifically, the generator can be described by the following equations:

$$\text{Initial state} \quad (\mathbf{c}_0, \mathbf{h}_0, \mathbf{u}_0) = (\mathbf{0}, \mathbf{0}, \mathbf{0}) \quad (3)$$

$$\text{Pre-process current canvas} \quad \mathbf{p}_l = \kappa(\mathbf{u}_l) \quad (4)$$

$$\text{State update} \quad (\mathbf{c}_{l+1}, \mathbf{h}_{l+1}) = C(\mathbf{E}, \mathbf{r}, \mathbf{c}_l, \mathbf{h}_l, \mathbf{p}_l) \quad (5)$$

$$\text{Canvas update} \quad \mathbf{u}_{l+1} = \mathbf{u}_l + \Delta(\mathbf{h}_{l+1}) \quad (6)$$

$$\text{Final output} \quad \mathbf{x} = \eta(\mathbf{u}_L), \quad (7)$$

with timestep  $l$  and final timestep  $L$ , LSTM output  $\mathbf{c}_l$  and cell  $\mathbf{h}_l$  states, the canvas  $\mathbf{u}_l$ , a downsampling network  $\kappa$ , the camera extrinsic parameters  $\mathbf{E}$ , an upsampling network  $\Delta$ , and a  $1 \times 1$  convolutional layer  $\eta$ . Consistent with [12], all up- and downsampling layers are convolutions of size  $4 \times 4$  with stride 4. To account for the higher resolution of the Shapenet v2 car and chair images, we added a further convolutional layer / transposed convolution where necessary.

**Training** On both the cars and chairs datasets, we trained for 180,000 iterations with a batch size of 140, taking approximately 6.5 days. For the lower-resolution Sheppard-Metzler objects, we trained for 160,000 iterations at a batch size of 192, or approximately 5 days.

**Testing** For novel view synthesis on the training set, the model receives as input the 15 nearest neighbors of the novel view in terms of cosine similarity. For two-shot reconstruction, the model receives as input whichever of the two reference views is closer to the novel view in terms of cosine similarity. For one-shot reconstruction, the model receives as input the single reference view.

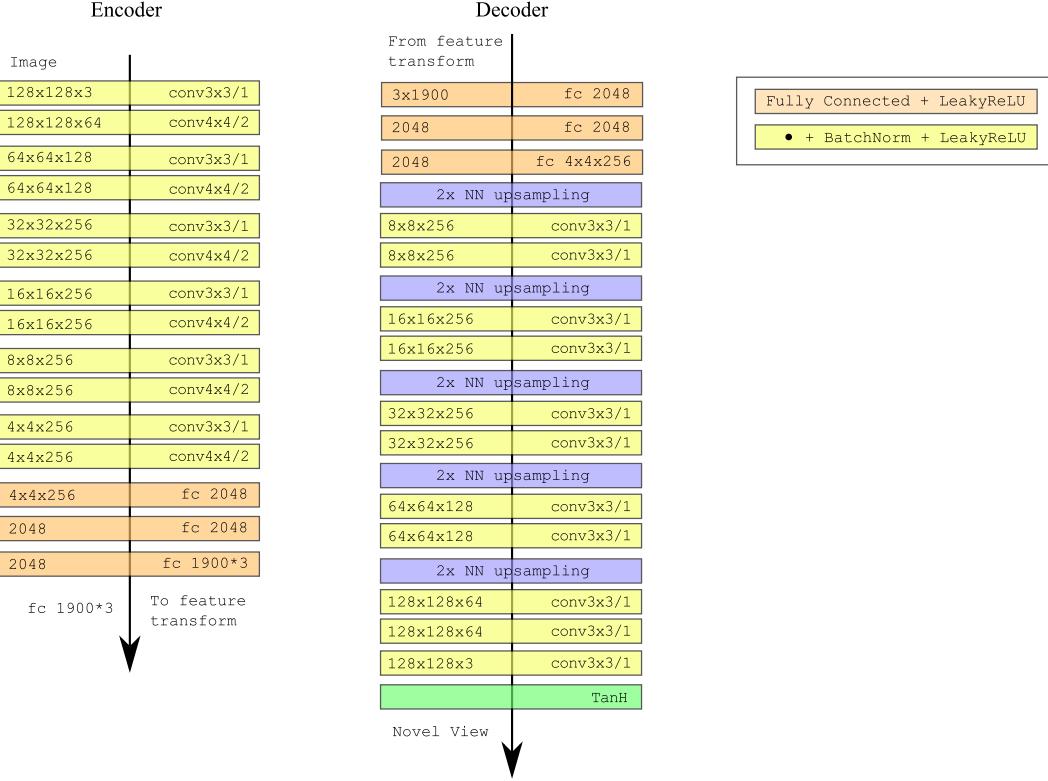


Figure 6: Architecture of the baseline method proposed in Worrall et al. [4].

## 5.2 Tatarchenko et al.

**Architecture** We implement the exact same architecture as described in [3], with approximately  $70 \cdot 10^6$  parameters.

**Training** For training, we choose the same hyperparameters as proposed in Tatarchenko et al. [3]. As we assume no knowledge of scene geometry, we do not supervise the model with a depth map. As we observed the model to overfit, we stopped training early based on model performance on the held-out, official Shapenet v2 validation set.

**Testing** For novel view synthesis on the training set, the model receives as input the nearest neighbor of the novel view in terms of cosine similarity. For two-shot reconstruction, the model receives as input whichever of the two reference views is closer to the novel view. Finally, for one-shot reconstruction, the model receives as input the single reference view.

## 5.3 Worrall et al.

**Architecture** Please see Fig. 6 for a visualization of the full architecture. The design choices in this architecture (nearest-neighbor upsampling, leaky ReLU activations, batch normalization) were made in accordance with Worrall et al. [4].

**Training** For training, we choose the same hyperparameters as proposed in Worrall et al. [4].

**Testing** For novel view synthesis on the training set, the model receives as input the nearest neighbor of the novel view in terms of cosine similarity. For two-shot reconstruction, the model receives as input whichever of the two reference views is closer to the novel view. Finally, for one-shot reconstruction, the model receives as input the single reference view.

## 6 Differentiable Ray-Marching in the context of classical renderers

The proposed neural ray-marcher is inspired by the classic sphere tracing algorithm [13]. Sphere tracing was originally developed to render scenes represented via analytical signed distance functions. It is defined by a special choice of the step length: each step has a length equal to the signed distance to the closest surface point of the scene. Since this distance is only zero on the surface of the scene, the algorithm takes non-zero steps until it has arrived at the surface, at which point no further steps are taken. A major downside of sphere-tracing is its weak convergence guarantee: Sphere tracing is only guaranteed to converge for an infinite number of steps. This is easy to see: For any fixed number of steps, we can construct a scene where a ray is parallel to a close surface (or falls through a slim tunnel) and eventually intersects a scene surface. For any constant number of steps, there exists a surface parallel to the ray that is so close that the ray will not reach the target surface. In classical sphere-tracing, this is circumvented by taking a large number of steps that generally take the intersection estimate within a small neighborhood of the scene surface – the color at this point is then simply defined as the color of the closest surface. However, this heuristic can still fail in constructed examples such as the one above. Extensions of sphere tracing propose heuristics to modifying the step length to speed up convergence [11]. The Ray-Marching LSTM instead has the ability to learn the step length. The key driver of computational and memory cost of the proposed rendering algorithm is the ray-marching itself: In every step of the ray-marcher, for every pixel, the scene representation  $\phi$  is evaluated. Each evaluation of  $\phi$  is a full forward pass through a multi-layer perceptron. See 3.2 for an exact analysis of memory and computational complexity of the different components.

Other classical rendering algorithms usually follow a different approach. In modern computer graphics, scenes are often represented via explicit, discretized surface primitives - such as is the case in meshes. This allows rendering via rasterization, where scene geometry is projected onto the image plane of a virtual camera in a single step. As a result, rasterization is computationally cheap, and has allowed for real-time rendering that has approached photo-realism in computer graphics.

However, the image formation model of rasterization is not appropriate to simulate physically accurate image formations that involve proper light transport, view-dependent effects, participating media, refraction, translucency etc. As a result, physics-based rendering usually uses ray-tracing algorithms, where for each pixel, a number of rays are traced from the camera via all possible paths to light sources through the scene. If the underlying scene representations are explicit, discrete representations – such as meshes – the intersection testing required is again cheap. Main drivers of computational complexity in such systems are then the number of rays that need to be traced to appropriately sample all paths to lights sources that contribute to the value of a single pixel.

In this context, the proposed ray-marcher can be thought of as a sphere-tracing-inspired ray-tracer for implicitly defined scene geometry. It does not currently model multi-bounce ray-tracing, but could potentially be extended in the future (see 8).

## 7 Trade-offs of the Pixel Generator vs. CNN-based renderers

As described in the main paper, the pixel generator comes with a guarantee of multi-view consistency compared to a 2D-CNN based rendering network. On the flip side, we cannot make use of progress in the design of novel CNN architectures that save memory by introducing resolution bottlenecks and skip connections, such as the U-Net [14]. This means that the pixel generator is comparably memory-hungry, as each layer operates on the full resolution of the image to be generated. Furthermore, CNNs have empirically been demonstrated to be able to generate high-frequency image detail easily. It is unclear what the limitations of the proposed pipeline are with respect to generating high-frequency textural detail. We note that the pixel generator is not a necessary component of SRNs, and can be replaced by a classic 2D-CNN based renderer, as we demonstrate in 2.

## 8 Future work

**Applications outside of vision.** SRNs have promising applications outside of vision. Neural scene representations are a core aspect of artificial intelligence, as they allow an agent to model its environment, navigate, and plan interactions. Thus, natural applications of SRNs lie in robotic manipulation or as the world model of an independent agent.

**Extending SRNs to other image formation models.** SRNs could be extended to other image formation models, such as computer tomography or magnetic resonance imaging. All that is required is a differentiable forward model of the image formation. The ray-marcher could be adapted accordingly to integrate features along a ray or to sample at pre-defined locations. For image formation models that observe scenes directly in 3D, the ray-marcher may be left out completely, and  $\phi$  may be sampled directly.

**Probabilistic formulation.** An interesting avenue of future work is to extend SRNs to a probabilistic model that can infer a probability distribution over feasible scenes consistent with a given set of observations. In the following, we formulate one such approach, very similar to the formulation of Kumar et al. [15], which is in turn based on the work of Eslami et al. [12]. Please note that this formulation is not experimentally verified in the context of SRNs and is described here purely to facilitate further research in this direction.

Formally, the model can be summarized as:

$$r_i = M(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i) \quad (8)$$

$$r = \sum_i r_i \quad (9)$$

$$z \sim P_\Theta(z|r) \quad (10)$$

$$\phi = \Psi(z) \quad (11)$$

$$\mathcal{I} = \Theta(\Phi_\phi, \mathbf{E}, \mathbf{K}) \quad (12)$$

We assume that we are given a set of instance datasets  $\mathcal{D} = \{\mathcal{C}_j\}_{j=1}^M$ , where each  $\mathcal{C}_j$  consists of tuples  $\{(\mathcal{I}_i, \mathbf{E}_i, \mathbf{K}_i)\}_{i=1}^N$ . For a single scene  $\mathcal{C}$  with  $n$  observations, we first replicate and concatenate the camera pose  $\mathbf{E}_i$  and intrinsic parameters  $\mathbf{K}_i$  of each observations to the image channels of the corresponding 2D image  $\mathcal{I}_i$ . Using a learned convolutional encoder  $M$ , we encode each of the  $n$  observations to a code vector  $r_i$ . These code vectors  $r_i$  are then summed to form a permutation-invariant representation of the scene  $r$ . Via an autoregressive DRAW model [16], we form a probability distribution  $P_\theta$  that is conditioned on the code vector  $r$  and sample latent variables  $z$ .  $z$  is decoded into the parameters of a scene representation network,  $\phi$ , via a hypernetwork  $\Psi(z) = \phi$ . Lastly, via our differentiable rendering function  $\Theta$ , we can render images  $\mathcal{I}$  from  $\Phi_\phi$  as described in the main paper. This allows to train the full model end-to-end given only 2D images and their camera parameters. We note that the resulting optimization problem is intractable and requires the optimization of an evidence lower bound via an approximate posterior, which we do not derive here – please refer to [15]. Similarly to [15], this formulation will lead to multi-view consistent renderings of each scene, as the scene representation  $\Phi$  stays constant across queries of  $\Theta$ .

**View- and lighting-dependent effects, translucency, and participating media.** Another exciting direction for future work is to model further aspects of realistic scenes. One such aspect is view- and lighting dependent effects, such as specularities. For fixed lighting, the pixel generator could receive as input the direction of the camera ray in world coordinates, and could thus reason about the view-dependent color of a surface. To model simple lighting-dependent effects, the pixel generator could further receive the light ray direction as an input (assuming no occlusions). Lastly, the proposed formulation could also be extended to model multiple ray bounces in a ray-casting framework. To model translucency and participating media, the ray-marcher could be extended to sum features along a ray instead of only sampling a feature at the final intersection estimate.

**Complex 3D scenes and compositionality.** While SRNs can represent room-scale scenes (see supplementary video), generalization across such complex, cluttered 3D environments is an open problem. To the best of our knowledge, it has not yet been demonstrated that low-dimensional embeddings are a feasible representation for photo-realistic, general 3D environments. Recent work in meta-learning could enable generalization across scenes without the limitation to a highly low-dimensional manifold [17].

## References

- [1] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “Deepvoxels: Learning persistent 3d feature embeddings,” in *Proc. CVPR*, 2019.

- [2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proc. CVPR*, 2017, pp. 5967–5976.
- [3] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Single-view to multi-view: Reconstructing unseen views with a convolutional network,” *CoRR abs/1511.06702*, vol. 1, no. 2, p. 2, 2015.
- [4] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, “Interpretable transformations with encoder-decoder networks,” in *Proc. ICCV*, vol. 4, 2017.
- [5] T. S. Cohen and M. Welling, “Transformation properties of learned visual representations,” *arXiv preprint arXiv:1412.7659*, 2014.
- [6] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.
- [7] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proc. CVPR*, 2015, pp. 1026–1034.
- [8] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Nießner, “Face2face: Real-time face capture and reenactment of rgb videos,” in *Proc. CVPR*, 2016, pp. 2387–2395.
- [9] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017.
- [10] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.
- [11] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, “Conditional image generation with pixelcnn decoders,” in *Proc. NIPS*, 2016.
- [12] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor *et al.*, “Neural scene representation and rendering,” *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.
- [13] J. C. Hart, “Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces,” *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.
- [14] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
- [15] A. Kumar, S. A. Eslami, D. Rezende, M. Garnelo, F. Viola, E. Lockhart, and M. Shanahan, “Consistent jumpy predictions for videos and scenes,” 2018.
- [16] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, “Draw: A recurrent neural network for image generation,” *arXiv preprint arXiv:1502.04623*, 2015.
- [17] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR.org, 2017, pp. 1126–1135.