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

Vincent Sitzmann Michael Zollhöfer Gordon Wetzstein

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

vsitzmann.github.io/srns/

Abstract

The advent of deep learning has given rise to neural scene representations – learned mathematical models of a 3D environment. However, many of these representations do not explicitly reason about geometry and thus do not account for the underlying 3D structure of the scene. In contrast, geometric deep learning has explored 3D-structure-aware representations of scene geometry, but requires explicit 3D supervision. 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 observations, without access to depth or geometry. 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.¹.

1 Introduction

Scene representations are mathematical models of a 3D environment. They are at the core of several unsolved problems in vision and artificial intelligence, such as reconstructing appearance and geometry from a few images, independent agent navigation and environment interaction, as well as multi-view consistent generative modeling. Tasks that are easy for humans to perform, such as inferring the shape, color, or material of an object from only a single image, remain challenging.

In recent years, many classic 3D scene representations, such as voxel grids [1–6], point clouds [7–10], or meshes [11] 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 and only sparsely sampling the underlying smooth surfaces of a scene. In contrast, a class of neural scene representations has recently emerged that do not, or only weakly, enforce 3D structure [12–14]. These approaches do not explicitly represent or reconstruct scene geometry. Multi-view geometry and projection operations required for rendering are performed by a black-box neural renderer, which is expected to learn these operations from data. As a result, they fail to capture 3D operations, such as camera translation and rotation with limited training data (see Sec. 4), and they lack guarantees on multi-view consistency of the rendered images. 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.

¹Please 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 SRNs currently do not model view-dependent effects and reconstruct shape and appearance in an entangled manner. Although we use SRNs in a deterministic framework, the formulation generalizes to modeling of uncertainty due to incomplete observations.

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 geometry representations to reason about scene geometry. Discretization-based techniques use voxel grids [3, 15–21], octree hierarchies [22–24], point clouds [7, 25, 26], multiplane images [27], patches [28], or meshes [11, 20, 29, 30]. Methods based on function spaces continuously represent space as the decision boundary of a learned binary classifier [31] or a continuous truncated signed distance field [32]. 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 efficiently encapsulates both scene geometry and appearance, and can be trained end-to-end via learned differentiable rendering with only 2D supervision in the image domain.

Neural Scene Representations Latent codes of autoencoders may be interpreted as a feature representation of the encoded scene. Rendering of novel views can be achieved by concatenating target pose and latent code [12] or performing view transformations directly in the latent space [14]. Generative Query Networks [13] introduce a powerful probabilistic reasoning framework that models uncertainty due to incomplete observations, but both the scene representation and the renderer are oblivious to the underlying 3D structure of our world. Some prior work infers voxel grid representations of 3D scenes from images [2, 4, 5] or uses them for 3D-structure-aware generative models [6, 33]. We demonstrate that models with unstructured scene representations fail to correctly perform viewpoint transformations in a regime of limited (but significant) data, such as the Shapenet v2 dataset [34]. Instead of a discrete representation, which limits achievable spatial resolution and does not smoothly parameterize scene surfaces, we propose a continuous scene representation. We focus on deterministic scene reconstruction, but note that the formulation of SRNs allows for a probabilistic extension in the future.

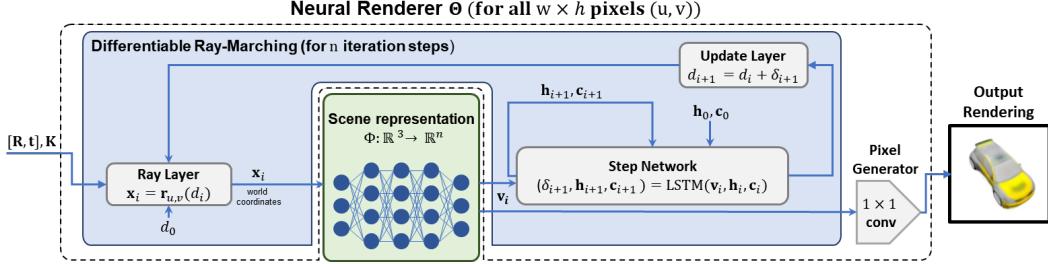


Figure 1: Overview: at the heart of SRNs lies a continuous, 3D-aware neural scene representation, Φ , which represents a scene as a function that maps (x, y, z) world coordinates to a feature representation of the scene at those coordinates (see Sec. 3.1). A neural renderer Θ , consisting of a learned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

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 (variational) auto-encoders [35, 36], invertible flows [37, 38], or autoregressive per-pixel models [39, 40]. In particular, generative adversarial networks [41–45] and their conditional variants [46–48] have recently achieved photo-realistic single-image generation. Compositional Pattern Producing Networks [49, 50] parameterize images as learned functions that map 2D image coordinates to color. Some approaches build on top of explicit spatial or perspective transformations in the networks [51–53]. Recently, following the spirit of “vision as inverse graphics” [54, 55], deep neural networks have been applied to the task of inverting graphics engines [56–60]. 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 [61], our goal is to distill this dataset of observations into a neural scene representation Φ that strictly enforces 3D structure and allows to generalize shape and appearance priors across scenes. In addition, we are interested in a rendering function Θ that allows us to render the scene represented by Φ from arbitrary viewpoints. In the following, we first formalize Φ and Θ and then discuss a framework for optimizing Φ , Θ 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 Φ 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 [1–6]. Point clouds [8–10] may contain points at any position in \mathbb{R}^3 , but only sparsely sample surface properties of a scene. In contrast, Φ 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 Φ as a multi-layer perceptron (MLP), and spatial resolution is thus limited by the representative power of the MLP.

In contrast to recent work on representing scenes as unstructured or weakly structured feature embeddings [12, 14, 13], Φ is explicitly aware of the 3D structure of scenes, as the input to Φ are world coordinates $(x, y, z) \in \mathbb{R}^3$. This allows interacting with Φ 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 Φ , we introduce a neural rendering algorithm Θ , that maps a scene representation Φ 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 Φ .

The key complication in rendering a scene represented by Φ 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 Φ 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

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 Ω .

Here, we take inspiration from the classic sphere tracing algorithm [62]. 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 [63]. We instead propose to *learn* the length of each step.

Specifically, we introduce a *ray marching long short-term memory (RM-LSTM)* [64], 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 δ 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 Φ 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

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  $\mathbf{r}_{u,v}(d_{max\_iter})$ 

```

\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 they are not required for training Φ .

3.2.2 Pixel Generator Architecture

The pixel generator takes as input the 2D feature map sampled from Φ 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 1×1 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 Θ operates independently on all pixels, allowing images to be generated with arbitrary resolutions and poses.

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 $(\mathcal{I}, \mathbf{E}, \mathbf{K})$ 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 Φ_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 ϕ_j live in a k -dimensional subspace of \mathbb{R}^l . Finally, we define a mapping

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

that maps a latent vector \mathbf{z}_j to the parameters ϕ_j of the corresponding Φ_j . We propose to parameterize Ψ as an MLP, with parameters ψ . This architecture was previously introduced as a Hypernetwork [65], a neural network that regresses the parameters of another neural network. We share the parameters of the rendering function Θ across scenes.

Finding latent codes \mathbf{z}_j . To find the latent code vectors \mathbf{z}_j , we follow an auto-decoder framework [32]. 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 Ψ and the neural renderer Θ . 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 [32] for additional details.

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 ψ of Ψ that maps latent vectors \mathbf{z}_j to the parameters of the respective scene

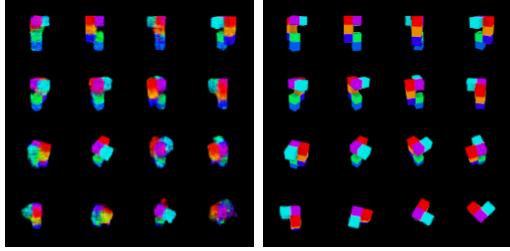


Figure 2: Shepard-Metzler object from 1k-object training set, 15 observations each. SRNs (right) outperform dGQN (left) on this small dataset.

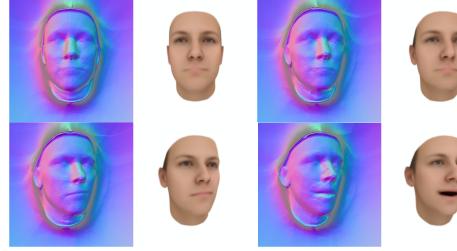


Figure 3: Non-rigid animation of a face. Note that mouth movement is directly reflected in the normal maps.

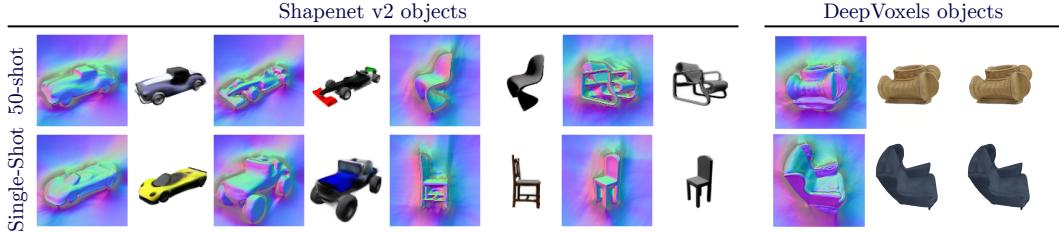


Figure 4: Normal maps for a selection of objects. We note that geometry is learned fully unsupervised and arises purely out of the perspective and multi-view geometry constraints on the image formation.

representation ϕ_j , the parameters θ of the neural rendering function Θ , as well as the latent codes \mathbf{z}_j themselves. We formulate this as an optimization problem with the following objective:

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

Where \mathcal{L}_{img} is an ℓ_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 ϕ of the MLP parameterization of Φ instead of the parameters ψ 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 θ as well as ψ , and estimate a new latent code $\hat{\mathbf{z}}$ by minimizing

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

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 [2].

Implementation Details Hyperparameters, computational complexity, and full network architectures for SRNs and all baselines are in the supplement. Code and datasets will be made 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 64×64 . We train



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.

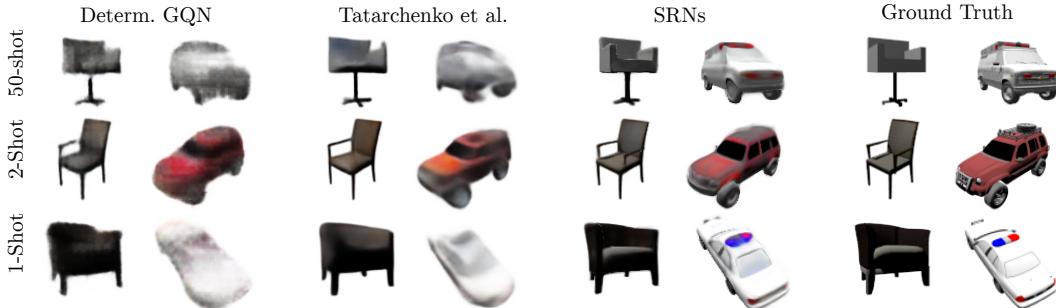


Figure 6: Qualitative comparison with Tatarchenko et al. [12] and the deterministic variant of the GQN [13], for novel view synthesis on the Shapenet v2 “cars” and “chairs” classes. We compare novel views for objects reconstructed from 50 observations in the training set (top row), two observations and 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.

both SRNs and a deterministic variant of the Generative Query Network [13] (dGQN, please see supplement for an extended discussion). 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 [34] 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 128×128 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 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.



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 [13] (dGQN), the model proposed by Tatarchenko et al. [12] (TCO), and the method proposed by Worrall et al. [14] (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 [12]	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 [14]	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 [13]	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	26.23 / 0.95	26.32 / 0.94	24.48 / 0.92	22.94 / 0.88	22.89 / 0.91	20.72 / 0.85

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 [66]. 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-shots, 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.

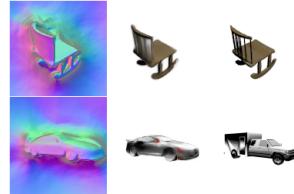


Figure 8: Failure cases.

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 3D supervision and can be trained with a set of posed 2D images of a scene. We demonstrate results for novel view synthesis, shape and appearance interpolation, and few-shot reconstruction.

References

- [1] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *Proc. IROS*, September 2015, p. 922 – 928.
- [2] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “Deepvoxels: Learning persistent 3d feature embeddings,” in *Proc. CVPR*, 2019.
- [3] A. Kar, C. Häne, and J. Malik, “Learning a multi-view stereo machine,” in *Proc. NIPS*, 2017, pp. 365–376.
- [4] H.-Y. F. Tung, R. Cheng, and K. Fragkiadaki, “Learning spatial common sense with geometry-aware recurrent networks,” *arXiv preprint arXiv:1901.00003*, 2018.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] E. Insafutdinov and A. Dosovitskiy, “Unsupervised learning of shape and pose with differentiable point clouds,” in *Proc. NIPS*, 2018, pp. 2802–2812.
- [9] M. Meshry, D. B. Goldman, S. Khamis, H. Hoppe, R. Pandey, N. Snavely, and R. Martin-Brualla, “Neural rerendering in the wild,” *arXiv preprint arXiv:1904.04290*, 2019.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, “Interpretable transformations with encoder-decoder networks,” in *Proc. ICCV*, vol. 4, 2017.
- [15] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, “Multi-view supervision for single-view reconstruction via differentiable ray consistency,” in *Proc. CVPR*, 2017.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] G. Riegler, A. O. Ulusoy, and A. Geiger, “Octnet: Learning deep 3d representations at high resolutions,” in *Proc. CVPR*, 2017.
- [23] 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.

- [24] C. Haene, S. Tulsiani, and J. Malik, “Hierarchical surface prediction,” *Proc. PAMI*, pp. 1–1, 2019.
- [25] 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.
- [26] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Multi-view 3d models from single images with a convolutional network,” in *Proc. ECCV*, 2016.
- [27] 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.
- [28] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, “Atlasnet: A papier-mâché approach to learning 3d surface generation,” *CoRR*, vol. abs/1802.05384, 2018.
- [29] H. Kato, Y. Ushiku, and T. Harada, “Neural 3d mesh renderer,” in *Proc. CVPR*, 2018, pp. 3907–3916.
- [30] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik, “Learning category-specific mesh reconstruction from image collections,” in *ECCV*, 2018.
- [31] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3d reconstruction in function space,” *arXiv preprint arXiv:1812.03828*, 2018.
- [32] 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.
- [33] T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y. Yang, “Hologan: Unsupervised learning of 3d representations from natural images,” *CoRR*, vol. abs/1904.01326, 2019.
- [34] 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.
- [35] G. E. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.
- [36] D. P. Kingma and M. Welling, “Auto-encoding variational bayes.” *CoRR*, 2013.
- [37] L. Dinh, D. Krueger, and Y. Bengio, “NICE: non-linear independent components estimation,” in *Proc. ICLR Workshops*, 2015.
- [38] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” in *NeurIPS*, 2018, pp. 10 236–10 245.
- [39] 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.
- [40] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.
- [41] 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.
- [42] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proc. ICML*, 2017.
- [43] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” in *Proc. ICLR*, 2018.
- [44] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, “Generative visual manipulation on the natural image manifold,” in *Proc. ECCV*, 2016.
- [45] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *Proc. ICLR*, 2016.
- [46] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” 2014, arXiv:1411.1784.
- [47] 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.
- [48] 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.

- [49] 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.
- [50] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah, “Differentiable image parameterizations,” *Distill*, vol. 3, no. 7, p. e12, 2018.
- [51] 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.
- [52] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, “Spatial transformer networks,” in *Proc. NIPS*, 2015.
- [53] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming auto-encoders,” in *Proc. ICANN*, 2011.
- [54] A. Yuille and D. Kersten, “Vision as Bayesian inference: analysis by synthesis?” *Trends in Cognitive Sciences*, vol. 10, pp. 301–308, 2006.
- [55] 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.
- [56] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, “Deep convolutional inverse graphics network,” in *Proc. NIPS*, 2015.
- [57] J. Yang, S. Reed, M.-H. Yang, and H. Lee, “Weakly-supervised disentangling with recurrent transformations for 3d view synthesis,” in *Proc. NIPS*, 2015.
- [58] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka, “Picture: A probabilistic programming language for scene perception,” in *Proc. CVPR*, 2015.
- [59] H. F. Tung, A. W. Harley, W. Seto, and K. Fragkiadaki, “Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision,” in *Proc. ICCV*.
- [60] Z. Shu, E. Yumer, S. Hadap, K. Sunkavalli, E. Shechtman, and D. Samaras, “Neural face editing with intrinsic image disentangling,” in *Proc. CVPR*, 2017.
- [61] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003.
- [62] 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.
- [63] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves *et al.*, “Conditional image generation with pixelcnn decoders,” in *Proc. NIPS*, 2016.
- [64] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [65] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” *arXiv preprint arXiv:1609.09106*, 2016.
- [66] 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.

Supplementary Material

Contents

1 Additional Results on Neural Ray Marching	2
2 Comparison to DeepVoxels	2
3 Reproducibility	4
3.1 Architecture Details	4
3.2 Time & Memory Complexity	4
3.3 Dataset Details	5
3.4 SRNs Training Details	5
3.4.1 General details	5
3.4.2 Per-experiment details	6
4 Relationship to per-pixel autoregressive methods	6
5 Baseline Discussions	6
5.1 Deterministic Variant of GQN	6
5.2 Tatarchenko et al.	7
5.3 Worrall et al.	7

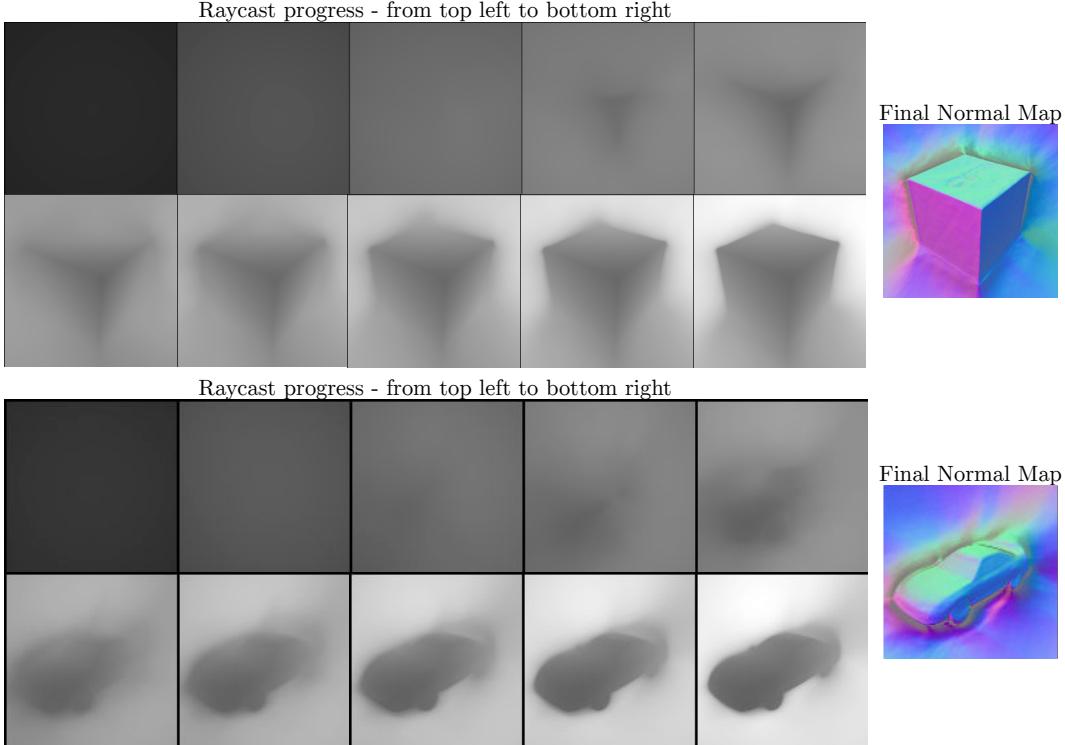


Figure 1: Visualizations of ray marching progress and the final normal map. Note that the uniformly colored background does not constrain the depth - as a result, the depth is unconstrained around the silhouette of the object. Since the final normal map visualizes surface detail much better, we only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

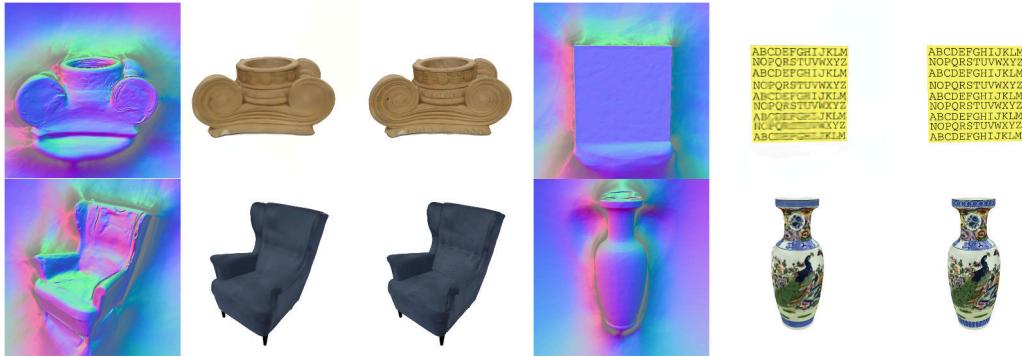


Figure 2: Qualitative results on DeepVoxels objects. For each object: Left: Normal map of reconstructed geometry. Center: SRNs output. Right: Ground Truth.

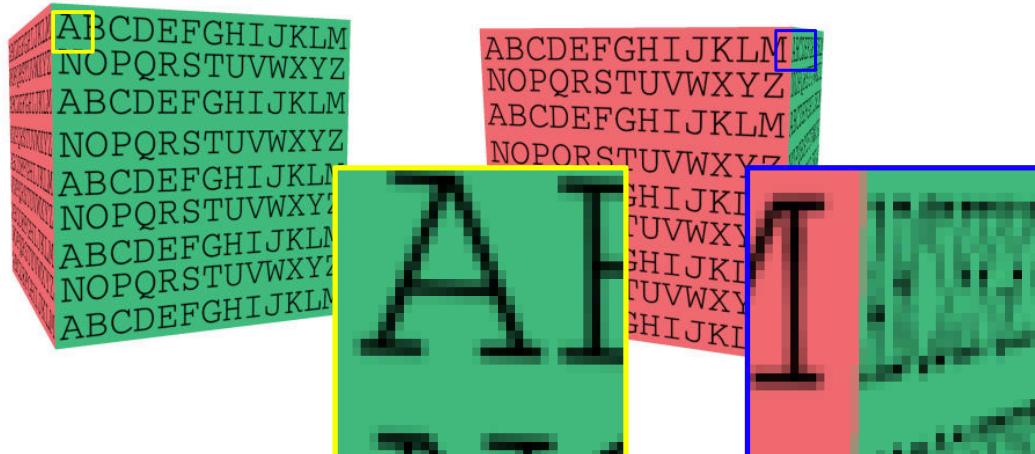


Figure 3: Undersampled letters on the side of the cube (ground truth images). Lines of letters are less than two pixels wide, leading to significant aliasing. Additionally, the 2D downsampling as described in [1] introduced blur that is not multi-view consistent.

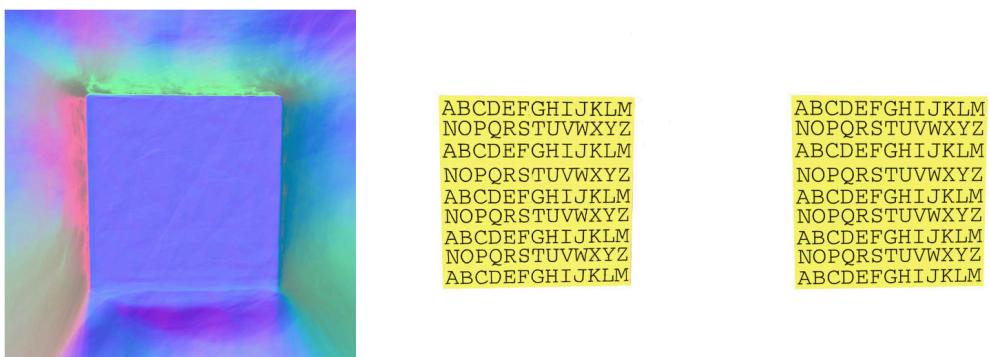


Figure 4: By using a U-Net renderer similar to [1], we can reconstruct the undersampled letters. In exchange, we lose the guarantee of multi-view consistency. Left: Reconstructed normal map. Center: SRNs output. Right: ground truth.

	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	33.03	0.97

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

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 Φ In all experiments, Φ 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 ϕ are the output of the hypernetwork Ψ . In the DeepVoxels comparison (see Sec.2), where a separate Φ is trained per scene, parameters of ϕ are directly initialized using the Kaiming Normal method [7].

Hypernetwork Ψ In generalization experiments, a hypernetwork Ψ maps a latent vector \mathbf{z}_j to the weights of the respective scene representation ϕ_j . Each layer of Φ 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 Φ 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 Φ Φ 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.

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

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 Φ repeatedly, it also scales linearly in the same parameters as Φ .

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 64×64 , 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. λ_{depth} is always set to 1×10^{-3} and λ_{latent} is set to 1. The ADAM learning rate is set to 4×10^{-4} if not reported otherwise.

Shepard-Metzler experiment We directly train our model on images of resolution 64×64 for 352 epochs.

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

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

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

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

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:

$$\begin{aligned} \text{Initial state} \quad & (\mathbf{c}_0, \mathbf{h}_0, \mathbf{u}_0) = (\mathbf{0}, \mathbf{0}, \mathbf{0}) & (3) \\ \text{Pre-process current canvas} \quad & \mathbf{p}_l = \kappa(\mathbf{u}_l) & (4) \\ \text{State update} \quad & (\mathbf{c}_{l+1}, \mathbf{h}_{l+1}) = C(\mathbf{E}, \mathbf{r}, \mathbf{c}_l, \mathbf{h}_l, \mathbf{p}_l) & (5) \\ \text{Canvas update} \quad & \mathbf{u}_{l+1} = \mathbf{u}_l + \Delta(\mathbf{h}_{l+1}) & (6) \\ \text{Final output} \quad & \mathbf{x} = \eta(\mathbf{u}_L), & (7) \end{aligned}$$

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 κ , the camera extrinsic parameters \mathbf{E} , an upsampling network Δ , and a 1×1 convolutional layer η . Consistent with [12], all up- and downsampling layers are convolutions of size 4×4 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 Shepard-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.

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. 5 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.

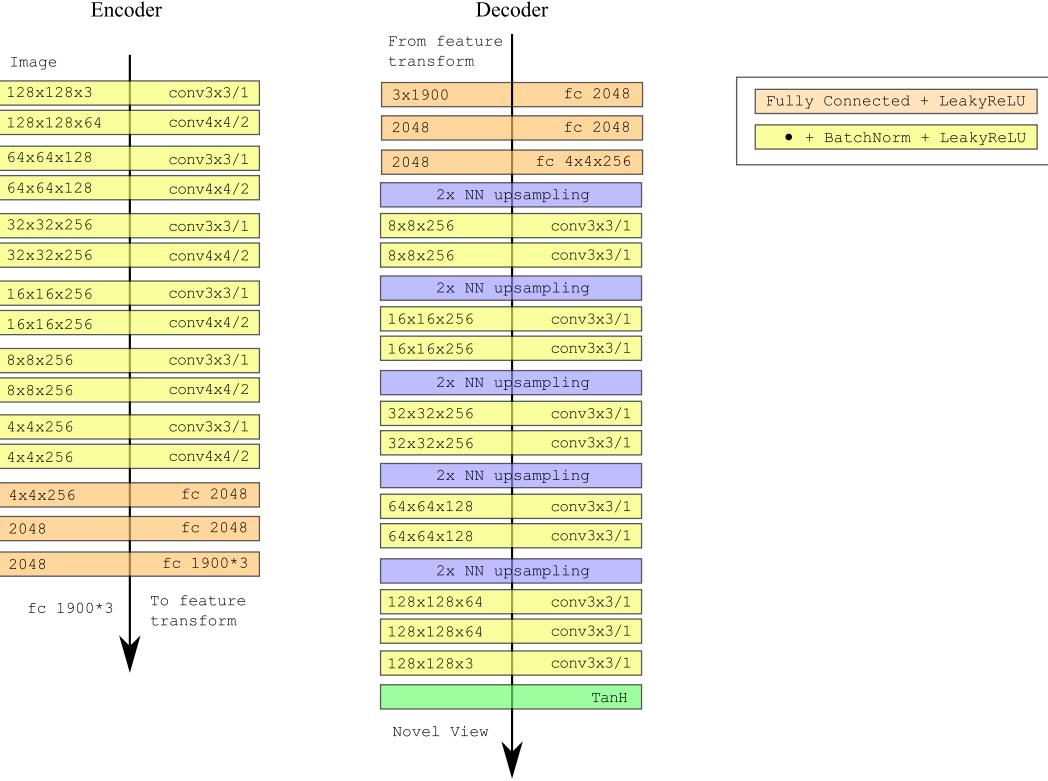


Figure 5: Architecture of the baseline method proposed in Worrall et al. [4].

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.