SynSin: End-to-end View Synthesis from a Single Image

Olivia Wiles[1]* Georgia Gkioxari[2] Richard Szeliski[3] Justin Johnson[2,4]

[1]University of Oxford [2]Facebook AI Research [3]Facebook [4]University of Michigan

Figure 1: **End-to-end view synthesis.** Given a *single* RGB image (red), SynSin generates images of the scene at new viewpoints (blue). SynSin predicts a 3D point cloud, which is projected onto new views using our differentiable renderer; the rendered point cloud is passed to a GAN to synthesise the output image. SynSin is trained end-to-end, without 3D supervision.
根据多帧预测3D点云，投影到新视角渲染新视图，渲染图进入GAN合成新视图，不需要3D监督

Abstract

View synthesis allows for the generation of new views of a scene given one or more images. This is challenging; it requires comprehensively understanding the 3D scene from images. As a result, current methods typically use multiple images, train on ground-truth depth, or are limited to synthetic data. We propose a novel end-to-end model for this task using a single image *at test time; it is trained on real images without any ground-truth 3D information. To this end, we introduce a novel differentiable point cloud renderer that is used to transform a latent 3D point cloud of features into the target view. The projected features are decoded by our refinement network to inpaint missing regions and generate a realistic output image. The 3D component inside of our generative model allows for interpretable manipulation of the latent feature space at test time,* e.g. *we can animate trajectories from a single image. Additionally, we can generate high resolution images and generalise to other input resolutions. We outperform baselines and prior work on the Matterport, Replica, and RealEstate10K datasets.*
由单张图生成动画轨迹

1. Introduction

Given an image of a scene, as in Fig. 1 (top-left), what would one see when turning left or walking forward? We can reason that the window and the wall will extend to the left and more chairs will appear to the right. The task of novel *view synthesis* addresses these questions: given a view of a scene, the aim is to generate images of the scene from new viewpoints. This task has wide applications in image editing, animating still photographs or viewing RGB images in 3D. To unlock these applications for any input image, our goal is to perform view synthesis in *complex, real-world scenes* using only *a single input image*.

View synthesis is challenging, as it requires comprehensive scene understanding. Specifically, successful view synthesis requires understanding both the **3D structure** and the **semantics** of the input image. Modelling 3D structure is important for capturing the relative motion of visible objects under a view transform. For example in Fig. 1 (bottom-left), the sink is closer than the shower and thus shifts more as we change viewpoints. Understanding semantics is necessary for synthesising plausible completions of partially visible objects, *e.g.* the chair in Fig. 1 (top-left).

One way to overcome these challenges is to relax the single-image constraint and use *multiple* views to reconstruct 3D scene geometry [12, 15, 51, 76]. This also simplifies semantic modelling, as fewer positions will be occluded from all views. Recent methods [56, 68, 74] can be extremely effective even for complex real-world scenes. However the assumption of multiple views severely limits their applicability, since the vast majority of images are not accompanied by views from other angles.

Project page: `www.robots.ox.ac.uk/~ow/synsin.html`.
*Work done during an internship at Facebook AI Research.

Another approach is to train a convolutional network to estimate depth from images [13, 33], enabling single-image view synthesis in realistic scenes [40]. Unfortunately this approach requires a training dataset of images with ground-truth depth. Worse, depth predictors may not generalise beyond the scene types on which they are trained (*e.g.* a network trained on indoor scenes will not work on outdoor images) so this approach can only perform view synthesis on scene types for which ground-truth depth can be obtained.

To overcome these shortcomings, there has been growing interest in view synthesis methods that do not use any 3D information during training. Instead, an end-to-end generative model with 3D-aware intermediate representations can be trained from image supervision alone. Existing methods have shown promise on synthetic scenes of single objects [31, 54, 55, 60, 67], but have been unable to scale to complex real-world scenes. In particular, several recent methods represent 3D structure using dense voxel grids of latent features [36, 54]. With voxels, the fidelity of 3D information that can be represented is tied to the voxel dimensions, thus limiting the output resolution. On the other hand, point clouds are more flexible, generalise naturally to varying resolutions and are more efficient.

In this paper we introduce SynSin, a model for view **syn**thesis from a **sin**gle image in complex real-world scenes. SynSin is an end-to-end model trained without any ground-truth 3D supervision. It represents 3D scene structure using a high-resolution point cloud of learned features, predicted from the input image using a pair of convolutional networks. To generate new views from the point cloud, we render it from the target view using a high-performance differentiable point cloud renderer. SynSin models scene semantics by building upon recent advances in generative models [3], and training adversarially against learned discriminators. Since all model components are differentiable, SynSin is trained end-to-end using image pairs and their relative camera poses; at test-time it receives only a single image and a target viewpoint.

We evaluate our approach on three complex real-world datasets: Matterport [4], RealEstate10K [74], and Replica [58]. All datasets include large angle changes and translations, increasing the difficulty of the task. We demonstrate that our approach generates high-quality images and outperforms baseline methods that use voxel-based 3D representations. We also show that our trained models can generalise at test-time to high-resolution output images, and even to new datasets with novel scene types.

2. Related work

Research into new view synthesis has a long history in computer vision. These works differ based on whether they use multiple images or a single image at test time and on whether they require annotated 3D or semantic information.

View synthesis from multiple images. If multiple images of a scene can be obtained, inferred 3D geometry can be used to reconstruct the scene and then generate new views. Traditionally, this was done using depth maps [5, 47] or multi-view geometry [11, 12, 15, 30, 51, 76].

In the learning era, DNNs can be used to learn depth. [1, 9, 23, 36, 38, 41] use a DNN to improve view synthesis from a set of noisy, incomplete, or inconsistent depth maps. Given two or more images of a scene within a small baseline, [16, 56, 57, 63, 68, 74] show impressive results at synthesising views within this narrow baseline. [35, 42, 54] learn an implicit voxel representation of one object given many training views and generate new views of that object at test time. [14] use no implicit 3D representation. Unlike these methods, we assume only one image at test time.

View synthesis from a single image using ground-truth depth or semantics. A second vein of work assumes a large dataset of images with corresponding ground-truth 3D and semantic information to train their 3D representation [40, 52, 62]. These methods are reliant on a large scale benchmark and corresponding annotation effort. The depth may be obtained using a depth or lidar camera [17, 28, 53] or SfM [33]; however, this is time-consuming and challenging, especially for outdoor scenes, often necessitating the use of synthetic environments. We aim to make predictions anywhere, *e.g.* the wood scene in Fig. 5, and in realistic settings, *without* 3D information or semantic labels.

View synthesis from a single image. DNNs can be used to learn view synthesis in an end-to-end fashion. One such line of work synthesises new views using purely image to image transformations [7, 31, 43, 59, 60, 75]. Later work performs 3D operations directly on the learned embedding [67] or interprets the latent space as an implicit surface [55]. However, these works consider synthetic datasets with a single object per image and train one model per object class. Most similar to ours is the recent work of [8]. However, they do not consider larger movements that lead to significant holes and dis-occlusions in the target image. They also consider a more constrained setup; they consider synthetic object classes and mostly forward motion in KITTI [17], whereas we use a variety of indoor and outdoor scenes.

Many works explore using a DNN to predict 3D object shapes [18, 20, 24, 26, 64, 69] or the depth of a scene given an image [6, 13, 33, 73]. These works focus on the quality of the 3D predictions as opposed to the view-synthesis task.

Generative models. We build on recent advances in generative models to produce high-quality images with DNNs [3, 19, 27, 39, 44]. In [3, 27], moving between the latent codes of different instances of an object class seemingly interpolates pose, but explicitly modifying pose is hard to control and evaluate. [39] allows for explicit pose control but not from a given image; they also use a voxel representation, which we find to be computationally limiting.

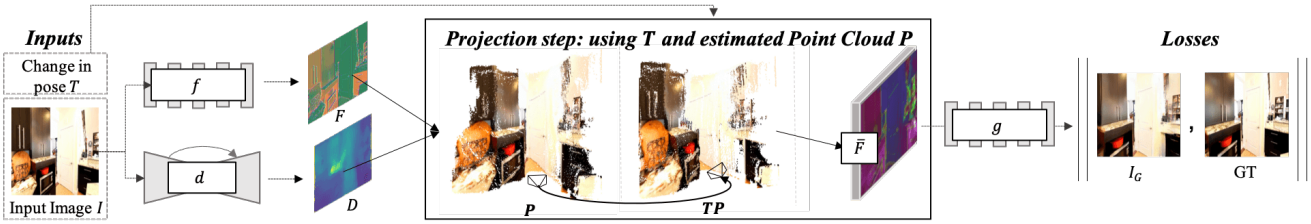


Figure 2: **Our end-to-end system.** The system takes as input an image I of a scene and change in pose T. The *spatial feature predictor* (f) learns a set of features F (visualised by projecting features using PCA to RGB) and the *depth regressor* (d) a depth map D. F are projected into 3D (the diagram shows RGB for clarity) to give a point cloud $\mathcal{P}$ of features. $\mathcal{P}$ is transformed according to T and rendered. The rendered features $\bar{F}$ are passed through the *refinement network* (g) to generate the final image I_G. I_G should match the target image, which we enforce using a set of discriminators and photometric losses.

3. Method

In this section, we introduce SynSin (Fig. 2) and in particular how we overcome the two main challenges of representing 3D scene structure and scene semantics. To represent the 3D scene structure, we project the image into a latent feature space which is in turn transformed using a differentiable point cloud renderer. This renderer injects a 3D prior into the network, as the predicted 3D structure must obey geometric principles. To satisfy the scene semantics, we frame the entire end-to-end system as a GAN and build on architectural innovations of recent state-of-the-art generative models.

SynSin takes an input image I and relative pose T. The input image is embedded to a feature space F via a *spatial feature predictor* (f), and a depth map D via a *depth regressor* (d). From F and D, a point cloud $\mathcal{P}$ is created which is rendered into the new view (*neural point cloud renderer*). The *refinement network* (g) refines the rendered features to give the final generated image I_G. At training time, we enforce that I_G should match the target image (*discriminator*).

3.1. Spatial feature and depth networks

f和d分别预测高维特征和深度

Two networks, f and d, are responsible for mapping the raw input image into a higher dimensional feature map and a depth map, respectively. The spatial feature network predicts feature maps at the same resolution as the original image. These feature maps should represent scene semantics, *i.e.* a higher-level representation than simply RGB colours. The depth network estimates the 3D structure of the input image at the same resolution. The depth does not have to be (nor would we expect it to be) perfectly accurate; however, it is explicitly learned in order to perform the task. The design for f and d follows standard architectures built for the two tasks respectively:

Spatial feature network f. We build on the BigGAN architecture [3] and use 8 ResNet blocks that maintain image resolution; the final block predicts a 64-dimensional feature for each pixel of the input image.

预测深度的分布，关于体进行渲染

Depth network d. We use a UNet [48] with 8 downsampling and upsampling layers to give a final prediction of the same spatial resolution as the input. This is followed by a sigmoid layer and a renormalisation step so the predicted depths fall within the per-dataset min and max values. Please refer to the supplement for the precise details.

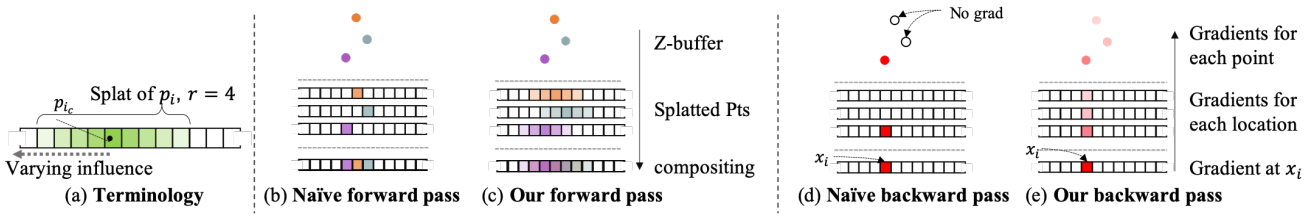
Unet预测深度

3.2. Neural point cloud renderer

We combine the spatial features F and predicted depths D to give a 3D point cloud of feature vectors $\mathcal{P}$. Given the input view transform T, we want to view this point cloud at the target viewpoint. This requires rendering the point cloud. Renderers are used extensively in graphics, as reviewed in [29, 49], but they usually focus on forward projection. Our 3D renderer is a component of an end-to-end system, which is jointly optimised, and so needs to allow for gradient propagation; we want to train for depth prediction *without* any 3D supervision but only with a loss on the final rendered image. Additionally, unlike traditional rendering pipelines, we are not rendering RGB colours but *features*.

Limitations of a naïve renderer. A naïve renderer projects 3D points p_i to one pixel or a small region – the *footprint* – in the new view. Points are sorted in depth using a z-buffer. For all points in the new view, the nearest point in depth (using the z-buffer) is chosen to colour that point. A non-differentiable renderer does not provide gradients with respect to the point cloud positions (needed to train our depth predictor) nor the feature vectors (needed to train our spatial feature network). Simply making the operations of a naïve renderer differentiable is problematic for two reasons (illustrated in Fig. 3). **(1) Small neighbourhoods:** each point projects to only one or a few pixels in the rendered view. In this case, there are only a few gradients for each point in the xy-plane of the rendered view; this drawback of *local* gradients is discussed in [25] in the context of bilinear samplers. **(2) The hard z-buffer:** each rendered pixel is only affected by the nearest point in the z-buffer (*e.g.* if a new pixel becomes closer in depth, the output will suddenly change).

传统投影方式只选择最近的点投影，不回传梯度；1、每个点只投影一个或少数像素，只有局部梯度；2、每个渲染的像素只受z最近的点影响



术语　　可微渲染=积分

Figure 3: Comparison of our rendering pipeline to a naïve version. Given a set of points ordered in a z-buffer, our renderer projects points to a region of radius r using α-compositing, not just the nearest point. When back-propagating through our renderer, gradients flow not just to the nearest point, but to all points in the z-buffer. (For simplicity we show 1D projections.)

Our solution. We propose a neural point cloud renderer in order to solve the prior two problems by softening the hard decisions, as in Fig. 3. This is inspired by [34], which introduces a differentiable renderer for meshes by similarly softening the hard rasterisation decisions and [77] which renderers point clouds by splatting points to a region and accumulating. First, to solve the issue of small neighbourhoods, we splat 3D points to a disk of varying influence controlled by hyperparameters r and M. Second, to solve the issue of the hard z-buffer, we accumulate the effects of the K nearest points, not just the nearest point, using a hyperparameter γ.

投影到各个区域
K邻域积分

Our renderer first projects $\mathcal{P}$ onto a 2D grid under the given transformation $\mathtt{T}$. A 3D point p_i is projected and splatted to a region with centre p_{i_c} and radius r. The influence of the 3D point p_i on a pixel l_{xy} is proportional to the Euclidean distance $d_2(\cdot,\cdot)$ from the centre of the region:

预测深度是一个分布，都是体，关于深度的体

$$\mathcal{N}(p_i, l_{xy}) = 0 \qquad \text{if} \quad d_2(p_{i_c}, l_{xy}) > r$$
$$\mathcal{N}(p_i, l_{xy}) = 1 - \frac{d_2(p_{i_c}, l_{xy})}{M} \qquad \text{otherwise.}$$

球外=0，球内与r负相关

Though $\mathcal{N}$ is not differentiable, we can approximate derivatives using the subderivative. r and M control the spread and fall-off of the influence of a 3D point.

The projected points are then accumulated in a z-buffer; they are sorted according to their distance d_i from the new camera and only the K nearest points kept for each pixel in the new view. The sorted points are accumulated using alpha over-compositing (where γ is a hyperparameter):

$$\rho_{i_{mn}} = \mathcal{N}(p_i, l_{mn}) \tag{1}$$
$$\bar{F}_{mn} = \sum_{i=1}^{K} \rho_{i_{mn}}^{\gamma} F_i \prod_{j=1}^{i-1} (1 - \rho_{j_{mn}}^{\gamma}), \tag{2}$$

where $\bar{F}$ is the projected feature map in the new view and F in the original view. γ controls the blending; if $\gamma = 0$, this is hard z-buffering. This setup is illustrated in Fig. 3.

Implementation. Our renderer must be high-performance, since we process batches of high-resolution point clouds during training. We implement our renderer using a sequence of custom CUDA kernels, building upon work on high-performance triangle rasterisation with CUDA [32]. We use a two-stage approach: in the first stage we break the output image into tiles, and determine the set of points whose footprint intersects each tile. In the second stage, we determine the K nearest points for each pixel in the output image, sorting points in depth using per-pixel priority queues in shared memory to reduce global memory traffic.

Other approaches. This method is related to the point cloud rasterisers of [1, 24, 70]. However, our renderer is a simpler than [70] and we apply it in an end-to-end framework. While [1] also renders point clouds of features, they only back-propagate to the feature vectors, not the 3D positions. [24] stores the predicted points in a voxel grid before performing the projection step; this limits the resolution.

Performance. On a single V100 GPU, rendering a batch of six point clouds with $512^2 = 262{,}144$ points each to a batch of six images of size 256×256 takes 36ms for the forward pass and 5ms for the backward pass. In contrast, converting the same point cloud to a 256^3 voxel grid using the implementation from [24] takes $\approx$ 1000ms for the forward pass and $\approx$ 2000ms for the backward pass.

3.3. Refinement module and discriminator

Even if the features are projected accurately, regions not visible in the input view will be empty in the target view. The refinement module should inpaint [2, 10] these missing regions in a semantically meaningful (*e.g.* missing portions of a couch should be filled in with similar texture) and geometrically accurate (*e.g.* straight lines should continue to be straight) manner. To solve this task, we take inspiration from recent generative models [3, 27, 44].

Deep networks have been previously applied to inpainting [46, 61, 66]. In a typical inpainting setup, we know a-priori which pixels are correct and which need to be synthesised. In our case, the refinement network should perform two tasks. First, it should inpaint regions with no projected features, *e.g.* regions on the image boundary or dis-occluded regions. The refinement module can discover these regions, as the features have values near zero. Second, the refinement module should correct local errors (*e.g.* noisy regions resulting from noisy depth).

refine网络：1、补全和扩展；2、纠正错误

Figure 4: Qualitative results on RealEstate for ours and baseline methods. Given the input view and the camera parameters, the methods are tasked to produce the target image. The red squares denote interesting differences between the methods. In the upper row, our model better recreates the true 3D; in the bottom row, our model is better able to preserve detail.

To build the refinement network, we use 8 ResNet [21] blocks, taking inspiration from [3]. Unlike [3], we aim to generate a new image conditioned on an input view not a random vector. Consequently, we find that it is important to maintain the image resolution as much as possible to obtain high quality results. We modify their ResNet block to create a downsampling block. The downsampling block is used to decrease the image resolution by two sizes before upsampling to the original image resolution. To model the ambiguity in the inpainting task, we use batch normalisation injected with noise [3]. We additionally apply spectral normalisation following each convolutional layer [71].

The GAN architecture and objective used is that of [65]. We use 2 multi-layer discriminators at a lower and higher resolution and a feature matching loss on the discriminator.

3.4. Training

Training objective. The network is trained with an L1 loss, content loss and discriminator loss between the generated and target image. The total loss is then $\mathcal{L} = \lambda_{GAN}\mathcal{L}_{GAN} + \lambda_{l1}\mathcal{L}_{l1} + \lambda_c\mathcal{L}_c$.

Training details. The models are trained with the Adam optimiser using a 0.01 learning rate for the discriminator, 0.0001 for the generator and momentum parameters (0, 0.9). $\lambda_{GAN} = 1$, $\lambda_c = 10$, $\lambda_{l1} = 1$. $\gamma = 1$, $r = 4$ pixels, $K = 128$, $W = H = 256$. The models are trained for 50K iterations. We implement our models in PyTorch [45]; they take 1-2 days to train on 3 Tesla V100 GPUs.

4. Experiments

We evaluate our approach on the task of view synthesis using novel real-word scenes. We validate our design choices in Section 4.3 by ablating our approach and comparing against competing end-to-end view synthesis pipelines. We also compare to other systems and find that our model performs better than one based on a trained depth predictor, which fails to generalise well to the new domain. We additionally evaluate SynSin's generalisation performance to novel domains (Section 4.3) as well as higher image resolutions (Section 4.4). Finally, we use SynSin to synthesise trajectories from an initial image in Section 4.6, demonstrating that it can be used for a walk-through application. Additional results are given in the supplement.

4.1. Experimental setup

Datasets. We focus on using realistic data of indoor and outdoor environments as opposed to synthetic objects.

The first framework we use is Habitat [50], which allows for testing in a variety of scanned indoor scenes. The Habitat framework can efficiently generate image and viewpoint pairs for an input scene. We use two sources of indoor scenes: Matterport3D [4], consisting of reconstructions of homes, and Replica [58], which consists of higher fidelity scans of indoor scenes. The Matterport3D dataset is divided at the scene level into train/val/test which contain 61/11/18 scenes. The Replica dataset is only used at evaluation time to test generalisability. Pairs of images are generated by randomly selecting a viewpoint in a scene and then randomly modifying the viewing angle in a range of $\pm 20^\circ$ in each Euclidean direction and the position within ± 0.32m.

The second dataset we use is RealEstate10K [74], which consists of videos of walkthroughs of properties and the corresponding camera parameters (intrinsic and extrinsic) obtained using SfM. The dataset contains both indoor and outdoor scenes. It is pre split into a disjoint set of train and test scenes; we subdivide train into a training and validation set to give approximately 57K/14K/7K scenes in train/val/test. The scenes in the test set are unseen. We sample viewpoints by selecting a reference video frame and then selecting a second video frame a maximum of 30 frames apart. In order to sample more challenging frames, we choose pairs

	Matterport [4]									RealEstate10K [74]			Replica [58]		
	PSNR ↑			SSIM ↑			Perc Sim ↓			PSNR ↑	SSIM ↑	Perc Sim ↓	PSNR ↑	SSIM ↑	Perc Sim ↓
	Both	InVis	Vis	Both	InVis	Vis	Both	InVis	Vis						
1. SynSin (small ft)	**21.14**	**20.19**	**21.84**	**0.71**	0.70	0.69	**1.68**	0.45	**0.98**	$21.10_{3.48}$	$0.73_{0.14}$	$1.34_{0.55}$	**22.36**	0.80	1.64
2. SynSin (hard z)	21.08	20.23	21.70	0.70	0.70	0.67	1.82	0.44	1.11	$21.40_{4.06}$	$0.70_{0.15}$	$1.45_{0.61}$	20.70	0.76	1.95
3. SynSin (rgb)	20.64	19.87	21.21	0.67	0.69	0.65	2.06	0.49	1.27	$20.92_{3.81}$	$0.68_{0.14}$	$1.67_{0.51}$	20.44	0.75	2.03
4. SynSin	20.91	19.80	21.62	**0.71**	**0.71**	**0.70**	**1.68**	**0.43**	**0.99**	$\mathbf{22.31}_{4.97}$	$\mathbf{0.74}_{0.16}$	$\mathbf{1.18}_{0.64}$	21.94	**0.81**	**1.55**
5. SynSin (w/ GT)	22.65	19.64	26.19	0.78	0.71	0.82	1.37	0.50	0.64	–	–	–	23.72	0.86	1.22
6. SynSin (sup. by GT)	21.59	20.32	22.46	0.72	0.71	0.71	1.60	0.43	0.92	–	–	–	22.54	0.80	1.55
7. Im2Im	15.87	16.20	15.97	0.53	0.60	0.48	2.99	0.58	2.05	$17.05_{4.78}$	$0.56_{0.18}$	$2.19_{1.22}$	17.42	0.66	2.29
8. Vox w/ UNet	18.52	17.85	19.05	0.57	0.57	0.57	2.98	0.77	1.96	$17.31_{2.63}$	$0.53_{0.15}$	$2.30_{0.40}$	18.69	0.71	2.68
9. Vox w/ ours	20.62	19.64	21.22	0.70	0.69	0.68	1.97	0.47	1.19	$21.88_{4.39}$	$0.71_{0.15}$	$1.30_{0.55}$	19.77	0.75	2.24

Table 1: Results on Matterport3D [4], RealEstate10K [74], and Replica [58]. ↑ denotes higher is better, ↓ lower is better. XX_{YY} denotes std dev. YY. The ablations demonstrate the utility of each aspect of our model. We outperform all baselines for both datasets and are nearly as good as a model supervised with depth (SynSin (sup. by GT)). We also perform best when considering regions visible (Vis) and not visible (InVis) in the input view.

with a change in angle of $> 5°$ and a change in position of greater than 0.15 if possible (see [74] for a discussion on metric scale). To report results, we randomly generate a set of 2000 pairs of images from the test set.

Metrics. Determining the similarity of images in a manner correlated with human judgement is challenging [72]. We report multiple metrics to obtain a more robust estimate of the relative quality of images. We report the PSNR, SSIM, and perceptual similarity of the images generated by the different models. Perceptual similarity has been recently demonstrated to be an effective method for comparing the similarity of images [72]. Finally, we validate that these metrics do indeed correlate with human judgement by performing a user study on Amazon Mechanical Turk (AMT).

4.2. Baselines

We first abate the need for a soft differentiable renderer by comparing to variants with a small footprint, hard z-buffering, and that directly project RGB values. These models use the same setup, training schedule, and sequence of input images/viewpoints as SynSin.

SynSin (small ft): We set $K = 128$ and $r = 0.5$ in our model to investigate the utility of a large footprint.

SynSin (hard z): We set $K = 1$ and $r = 4$ in our model to investigate the utility of the soft z-buffer.

SynSin (rgb): We project RGB values not features.

SynSin does not assume ground-truth depth at test time; the depth predictor is trained end-to-end for the given task. We investigate the impact of ground-truth (GT) depth by reporting two variants of our model. These models act as upper bounds and can only be trained on Matterport3D (not RealEstate10K), as they use true depth information.

SynSin (w/ GT): The true depth is used as D.

SynSin (sup. by GT): D is supervised by the true depth. (In all other cases SynSin's D is learned with *no* supervision).

We evaluate our 3D representation by comparing to a method that uses no 3D and one that uses voxels. As no methods exist for the challenging datasets we consider, we re-implement the baselines for a fair comparison. The baselines use the same setup, training schedule, and sequence of input images/viewpoints as SynSin.

Im2Im: This baseline evaluates an image-to-image method; we re-implement [75]. [75] only considered a set of discretised rotations about the azimuth and a smaller set of rotations in elevation. However, the changes in viewpoint in our datasets arise from rotating continuously in any direction and translating in 3D. We modify their method to allow for these more complex transformations.

Vox: This baseline swaps our implicit 3D representation for a voxel based representation. The model is based on that of [54]. However, [54] trains one model per object, so their model effectively learns to interpolate between the $>$100 training views unlike our model, which extrapolates to new real-world test scenes given a *single* input view. We consider two variants: **Vox w/ UNet** uses the UNet encoder/decoder of [54] whereas **Vox w/ ours** uses a similar ResNet encoder/decoder setup to SynSin. This comparison evaluates our 3D approach as opposed to a voxel based one as well as whether our encoder/decoder setup is preferable.

Finally, we compare SynSin to existing pipelines that perform view synthesis. These systems make different assumptions and follow different approaches. This comparison validates our use of a learned end-to-end system.

StereoMag [74]: This system takes two images as input at test time. Assuming two input views simplifies the problem of 3D understanding compared to our work, which estimates 3D from a single view.

3D View: This system trains a single-image depth predictor on images with ground-truth depth (*e.g.* MegaDepth [33]). Predicted depths are used to convert the input image to a textured 3D mesh, which is extended in space near occlusion boundaries using isotropic colour diffusion [22]. Finally the mesh is rendered from the target view. The approach is similar to 3D Photos [37].

System comparison on RealEstate10K [74]			
	PSNR ↑	SSIM ↑	Perc Sim ↓
SynSin	$22.31_{4.97}$	$0.74_{0.16}$	$1.18_{0.64}$
3DView	$21.88_{8.43}$	$0.66_{0.22}$	$1.52_{1.03}$
StereoMag [74]	$25.34_{9.48}$	$0.82_{0.13}$	$1.19_{0.77}$

Table 2: SynSin performs better than a system trained with GT depth (3DView) and approaches the performance of [74], which uses 2 input views at test time.

	Generalisation to higher res.		
	PSNR ↑	SSIM ↑	Perc Sim ↓
SynSin	$\mathbf{22.06}_{6.30}$	$\mathbf{0.72}_{0.18}$	$\mathbf{1.00}_{0.65}$
Vox w/ ours	$18.82_{2.52}$	$0.61_{0.14}$	$2.47_{0.36}$

Table 3: Results when applying models trained on 256×256 images to 512×512 images.

AMT User Study			
	Ours	Vox w/ ours	Neither
E-O	**68.7**	31.3	–
E-O-N	**55.6**	27.3	17.2

Table 4: % of videos chosen as most realistic. In *E-O*, users choose the better method; in *E-O-N*, users can say neither is better.

Figure 5: System comparisons on RealEstate10K, illustrating failure cases. Note StereoMag [74] uses two input images (second is shown as an inset). Unlike [74] we inpaint missing regions (bottom row); [74] fails to model the left region and cannot inpaint the missing region. 3DView uses a model pretrained for depth, causing their system to produce inaccurate results in some cases (*e.g.* the bed in the top row).

4.3. Comparisons with other methods

Results on Matterport3D and RealEstate10K. We train our models, ablations, and baselines on these datasets.

To better analyse the results, we compare models on how well they understand the 3D scene structure and the scene semantics (discussed in Section 1). To achieve this, we report metrics on the final prediction (Both) but also on the regions of the target image that are visible (Vis) and not visible (InVis) in the input image. (Vis) evaluates the quality of the learned 3D scene structure, as it can be largely solved by accurate depth prediction. (InVis) evaluates the quality of a model's understanding of scene semantics; it requires a holistic understanding of semantic and geometric properties to reasonably in-paint missing regions. In order to determine the (Vis) and (InVis) regions, we use the GT depth in the input view to obtain a binary mask of which pixels are visible in the target image. This is only possible on Matterport3D (RealEstate10K does not have GT depth).

Table 1 and Fig. 4 report results on Matterport3D and RealEstate10K. On both datasets, we perform better than the baselines on all metrics and under all conditions, demonstrating the utility of both our 3D representation and our inpainting module. These results demonstrate that the differentiable renderer is important for training the depth model (rows 1-4). Our encoder decoder setup is shown to be important, as it improves the baseline's performance significantly (rows 8-9). Qualitatively, our model preserves fine detail and predicts 3D structure better than the baselines.

System comparison on RealEstate10K. We compare our system to the 3DView and StereoMag [74] in Table 2 and Fig. 5. Our model performs better than 3DView despite their method having been trained with hundreds of thousands of depth images. We hypothesise that this gap in performance is due to the 3DView's depth prediction not generalising well; their dataset consists of images of mostly close ups of objects whereas ours consists of scenes taken inside or outdoors. This baseline demonstrates that using an explicit 3D representation is problematic when the test domain differs from the training domain, as the depth predictor cannot generalise. Finally, our method of inpainting is better than that of 3DView, which produces a blurry result. [74] does not inpaint unseen regions in the generated image.

Comparison with upper bounds. We compare our model to SynSin (w/ GT) and SynSin (sup. with GT) in Table 1. These models either use GT depth or are supervised by GT depth; they are upper bounds of performance. While there is a performance gap between SynSin and SynSin (w/ GT) under the (Vis) condition, this gap shrinks for the (InVis) condition. Interestingly, SynSin trained with *no* depth supervision performs nearly as well as SynSin (sup. with GT) under both the (Vis) and (InVis) conditions; our model also generalises better to the Replica dataset. This experiment demonstrates that having true depth during training does not necessarily give a large boost in a downstream task and could hurt generalisation performance. It validates our decision to use an end-to-end system (as opposed to using depth estimated from a self-supervised method).

Generalisation to Replica. Given the models trained on Matterport3D, we evaluate generalisation performance (with no further fine-tuning) on Replica in Table 1. Replica contains additional types of rooms (*e.g.* office and hotel rooms) and is higher quality than Matterport (it has fewer geometric and lighting artefacts and more complex textures). SynSin generalises better to this unseen dataset; qualitatively, SynSin seems to introduce fewer artefacts (Fig. 6).

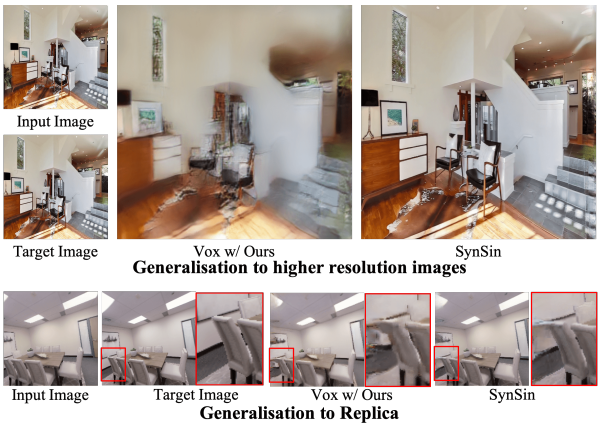


Figure 6: Comparison of SynSin against the baseline, Vox w/ ours, at generalising to higher res 512×512 images and Replica [58]. Ours generalises better with fewer artefacts.

4.4. Generalisation to higher resolution images

We also evaluate generalisation to higher image resolutions in Table 3 and Fig. 6. SynSin can be applied to higher resolution images without any further training. The ability to generalise to higher resolutions is due to the flexible 3D representation in our approach: the networks are fully convolutional and the 3D point cloud can be sampled at any resolution to maintain the resolution of the features. As a result, it is straightforward at test time to apply a network trained on a smaller image size (*e.g.* 256×256) to one of a different size (*e.g.* 512×512). Unlike our approach, the voxel baseline suffers a dramatic performance drop when applied to a higher resolution image. This drop in performance is presumably a result of the heavy downsampling and imprecision resulting from representing the world as a coarse voxel grid.

4.5. Depth predictions

We evaluate the quality of the learned 3D representation qualitatively in Fig. 7 for SynSin trained on RealEstate10K. We note that the accuracy of the depth prediction only matters in so far as it improves results on the view synthesis task. However, we hypothesise that the quality of the generated images and predicted depth maps are correlated, so looking at the quality of the depth maps should give some insight into the quality of the learned models. The depth map predicted by our method is higher resolution and more realistic than the depth map predicted by the baseline methods. Additionally, our differentiable point cloud renderer appears to improve the depth quality over using a hard z-buffer or a smaller footprint. However, we note that small objects and finer details are not accurately recreated. This is probably because these structures have a limited impact on the generated images.

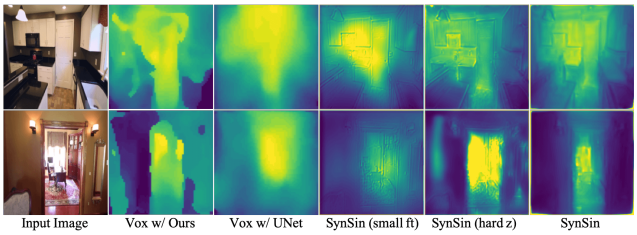


Figure 7: Recovered depth predictions for both our method and the baselines. The baselines predict a less accurate and coarser depth. Using a smaller radius or hard z-buffer produces qualitatively similar or worse depth maps. 合成深度较好

4.6. User study: Animating still images

Finally, we task SynSin to synthesise images along a trajectory. Given an initial input frame from a video in RealEstate10K, SynSin generates images at the camera position of the 30 subsequent frames. While changes are hard to see in a figure (*e.g.* Fig. 1), the supplementary videos clearly show smooth motion and 3D effects. These demonstrate that SynSin can generate reasonable videos despite being trained purely on images. To evaluate the quality of the generated videos, we perform an AMT user study.

We randomly choose 100 trajectories and generate videos using SynSin and the Vox w/ ours baseline. Five users are asked to rate which method's video is most realistic. For each video, we take the majority vote to determine the best video. We report the percentage of times the users choose a given method in Table 4.

Either-or setup (E-O): Users rate whether the baseline or our generated video is more realistic.

Either-or-neither setup (E-O-N): Users rate whether the baseline or our generated video is more realistic or whether they are equally realistic/unrealistic (*neither*). When taking the majority vote, if their is no majority, *neither* video is said to be more / less realistic

In both cases, users prefer our method, presumably because our videos have smoother motion and fewer artefacts.

5. Conclusion

We introduced SynSin, an end-to-end model for performing single image view synthesis. At the heart of our system are two key components: *first* a differentiable neural point cloud renderer, and *second* a generative refinement module. We verified that our approach can be learned end-to-end on multiple realistic datasets, generalises to unseen scenes, can be applied directly to higher image resolutions, and can be used to generate reasonable videos along a given trajectory. While we have introduced SynSin in the context of view synthesis, we note that using a neural point cloud renderer within a generative model has applications in other tasks.

Acknowledgements The authors thank Johannes Kopf for sharing code, Manolis Savva and Erik Wijmans for help with the Habitat dataset, and Sebastien Ehrhardt, Oliver Groth, and Weidi Xie for feedback on paper drafts.

References

[1] Kara-Ali Aliev, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. *arXiv preprint arXiv:1906.08240*, 2019.

[2] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proc. ACM SIGGRAPH*, 2000.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *Proc. ICLR*, 2019.

[4] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. Matterport3D dataset available at `https://niessner.github.io/Matterport/`.

[5] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 2013.

[6] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. In *NeurIPS*, 2016.

[7] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016.

[8] Xu Chen, Jie Song, and Otmar Hilliges. Monocular neural image based rendering with continuous view control. *Proc. ICCV*, 2019.

[9] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. *Proc. ICCV*, 2019.

[10] A. Criminisi, P. Pérez, and T. Kentaro. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 2004.

[11] Paul Debevec, Yizhou Yu, and George Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques*. 1998.

[12] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image- based approach. In *Proc. ACM SIGGRAPH*, pages 11–20, 1996.

[13] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NeurIPS*, 2014.

[14] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, and Karol Gregor. Neural scene representation and rendering. *Science*, 360(6394), 2018.

[15] A. W. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. *IJCV*, 63(2):141–151, 2005.

[16] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proc. CVPR*, 2019.

[17] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[18] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. *Proc. ICCV*, 2019.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[20] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. *Proc. CVPR*, 2018.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

[22] Peter Hedman and Johannes Kopf. Instant 3d photography. *ACM Transactions on Graphics (TOG)*, 2018.

[23] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 2018.

[24] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *NeurIPS*, 2018.

[25] Wei Jiang, Weiwei Sun, Andrea Tagliasacchi, Eduard Trulls, and Kwang Moo Yi. Linearized multi-sampling for differentiable image transformation. *Proc. ICCV*, 2019.

[26] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proc. ECCV*, 2018.

[27] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2019.

[28] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (TOG)*, 2017.

[29] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 2004.

[30] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. Image-based rendering in the gradient domain. *ACM Transactions on Graphics (TOG)*, 2013.

[31] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NeurIPS*, 2015.

[32] Samuli Laine and Tero Karras. High-performance software rasterization on gpus. In *Proc. ACM SIGGRAPH Symposium on High Performance Graphics.*, 2011.

[33] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proc. CVPR*, 2018.

[34] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised

single-view mesh reconstruction. *Proc. ICCV*, 2019.

[35] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 2019.

[36] Ricardo Martin-Brualla, Rohit Pandey, Shuoran Yang, Pavel Pidlypenskyi, Jonathan Taylor, Julien Valentin, Sameh Khamis, Philip Davidson, Anastasia Tkach, Peter Lincoln, et al. Lookingood: enhancing performance capture with real-time neural re-rendering. *ACM Transactions on Graphics (TOG)*, 2018.

[37] Kevin Matzen, Matthew Yu, Jonathan Lehman, Peizhao Zhang, Jan-Michael Frahm, Peter Vajda, Johannes Kopf, and Matt Uyttendaele. Powered by AI: Turning any 2D photo into 3D using convolutional neural nets. https://ai.facebook.com/blog/-powered-by-ai-turning-any-2d-photo-into-3d-using-convolutional-neural-nets/, Feb 2020.

[38] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *Proc. CVPR*, 2019.

[39] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3d representations from natural images. In *Proc. ICCV*, 2019.

[40] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3D Ken Burns effect from a single image. *ACM Transactions on Graphics (TOG)*, 2019.

[41] David Novotny, Ben Graham, and Jeremy Reizenstein. PerspectiveNet: A scene-consistent image generator for new view synthesis in real indoor environments. In *NeurIPS*, 2019.

[42] Kyle Olszewski, Sergey Tulyakov, Oliver Woodford, Hao Li, and Linjie Luo. Transformable bottleneck networks. In *Proc. ICCV*, 2019.

[43] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C Berg. Transformation-grounded image generation network for novel 3D view synthesis. In *Proc. CVPR*, 2017.

[44] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proc. CVPR*, 2019.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[46] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proc. CVPR*, pages 2536–2544, 2016.

[47] Eric Penner and Li Zhang. Soft 3D reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 2017.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. MICCAI*, pages 234–241. Springer, 2015.

[49] Miguel Sainz and Renato Pajarola. Point-based rendering techniques. *Computers & Graphics*, 2004.

[50] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proc. ICCV*, 2019.

[51] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. CVPR*, 2006.

[52] Daeyun Shin, Zhile Ren, Erik Sudderth, and Charless Fowlkes. 3D scene reconstruction with multi-layer depth and epipolar transformers. In *Proc. ICCV*, 2019.

[53] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *Proc. ECCV*, 2012.

[54] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. DeepVoxels: Learning persistent 3D feature embeddings. In *Proc. CVPR*, 2019.

[55] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019.

[56] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proc. CVPR*, 2019.

[57] Pratul P Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. Learning to synthesize a 4D rgbd light field from a single image. In *Proc. ICCV*, 2017.

[58] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[59] Shao-Hua Sun, Minyoung Huh, Yuan-Hong Liao, Ning Zhang, and Joseph J Lim. Multi-view to novel view: Synthesizing novel views with self-learned confidence. In *Proc. ECCV*, 2018.

[60] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3D models from single images with a convolutional network. In *Proc. ECCV*, 2016.

[61] Piotr Teterwak, Aaron Sarna, Dilip Krishnan, Aaron Maschinot, David Belanger, Ce Liu, and William T Freeman. Boundless: Generative adversarial networks for image extension. In *Proc. ICCV*, 2019.

[62] Shubham Tulsiani, Saurabh Gupta, David F Fouhey, Alexei A Efros, and Jitendra Malik. Factoring shape, pose, and layout from the 2D image of a 3D scene. In *Proc. CVPR*, 2018.

[63] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3D scene inference via view synthesis. In *Proc. ECCV*, 2018.

[64] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and

Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. CVPR*, 2017.

[65] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional GANs. In *Proc. CVPR*, 2018.

[66] Yi Wang, Xin Tao, Xiaoyong Shen, and Jiaya Jia. Wide-context semantic image extrapolation. In *Proc. CVPR*, 2019.

[67] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Interpretable transformations with encoder-decoder networks. In *Proc. ICCV*, 2017.

[68] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep view synthesis from sparse photometric images. *ACM Transactions on Graphics (TOG)*, 2019.

[69] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NeurIPS*, 2016.

[70] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 2019.

[71] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. 2019.

[72] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018.

[73] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proc. CVPR*, 2017.

[74] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Transactions on Graphics (TOG)*, 2018.

[75] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *Proc. ECCV*, 2016.

[76] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM transactions on graphics (TOG)*, 2004.

[77] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proc. ACM SIGGRAPH*, 2001.

SynSin: Appendix

We give additional results in Section A, additional architectural details in Section B, additional information about baselines in Section C, and finally information about datasets in Section D. Finally, we discuss some choices that did and did not work in Section E.

A. Additional experimental results

Results on KITTI [17]. We evaluated our model on the KITTI dataset in order to compare with [8]. We trained our model on the KITTI dataset and compared with their pretrained model on a held-out set in Table 5. We achieve similar or better results on all metrics. Additionally, because [8] resamples the input image, it cannot generate pixels unseen in the original view. As shown in Fig. 8 (rows 1,2,6), this causes severe artefacts for backward motion.

We additionally show some failure cases for both methods when the viewpoint change is much larger than the average viewpoint change seen at test time in the bottom two rows.

Additional qualitative results. We give additional qualitative results on RealEstate10K (Fig. 9-10), Replica (Fig. 11), and Matterport3D (Fig. 12). The supplementary video shows sample videos of a model generating images along a given trajectory. We compare SynSin to the baseline (Vox w/ ours); SynSin has smoother motion with fewer artefacts. We also visualise additional depth prediction results in Fig. 13-14.

Comparison on KITTI [17]			
	PSNR ↑	SSIM ↑	Perc Sim ↓
SynSin	16.70	0.52	**2.07**
ContView [8]	**16.90**	**0.54**	2.21

Table 5: Comparison on KITTI to [8]. ↑ denotes higher is better, ↓ lower is better.

Figure 8: Qualitative results on KITTI [74] comparing SynSin to [8]. The bottom two rows demonstrate failure cases due to large viewpoint change. Zoom in for details.

Figure 9: Additional results on RealEstate10K [74]. Zoom in for details.

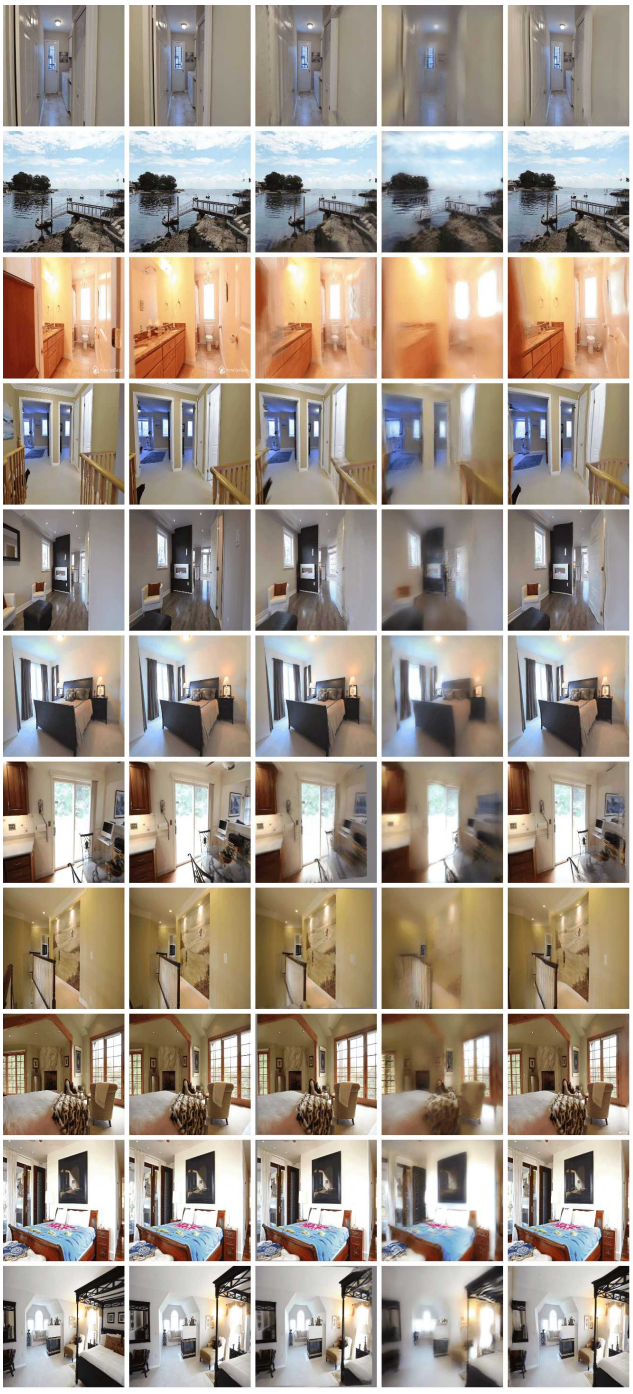


Figure 10: Additional results on RealEstate10K [74]. Zoom in for details.

Figure 11: Additional results on Replica [58]. Zoom in for details.

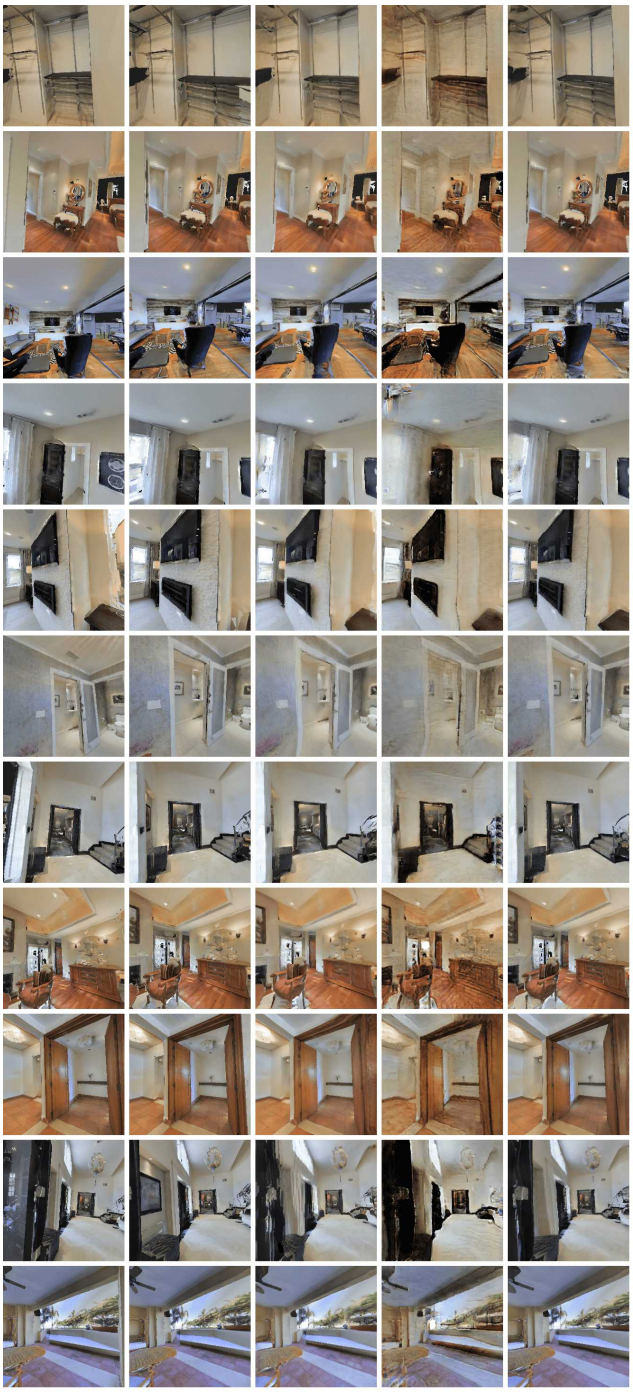


Figure 12: Additional results on Matterport3D [4]. Zoom in for details.

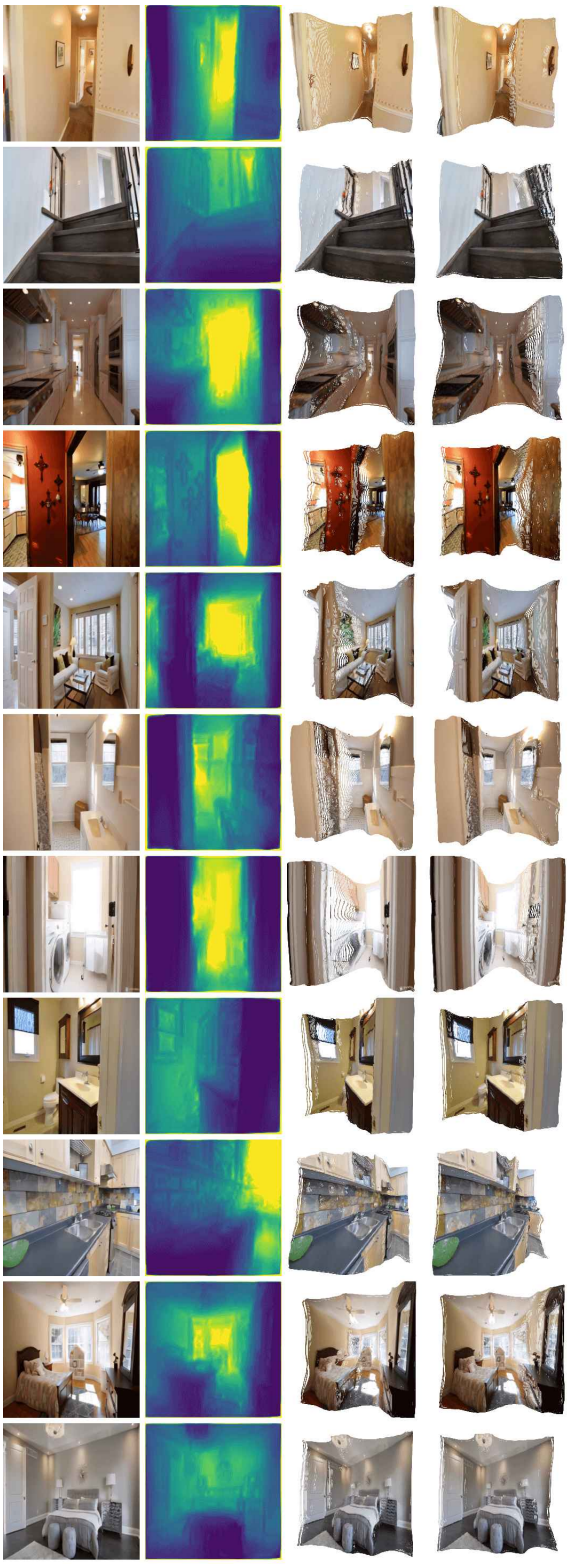


Figure 13: Additional depth predictions on RealEstate10K [74]. We also visualise the point cloud (PC) and the rotated point cloud at $-45°$. (Note that the point cloud in the model is actually a point cloud of features, not RGB values.)

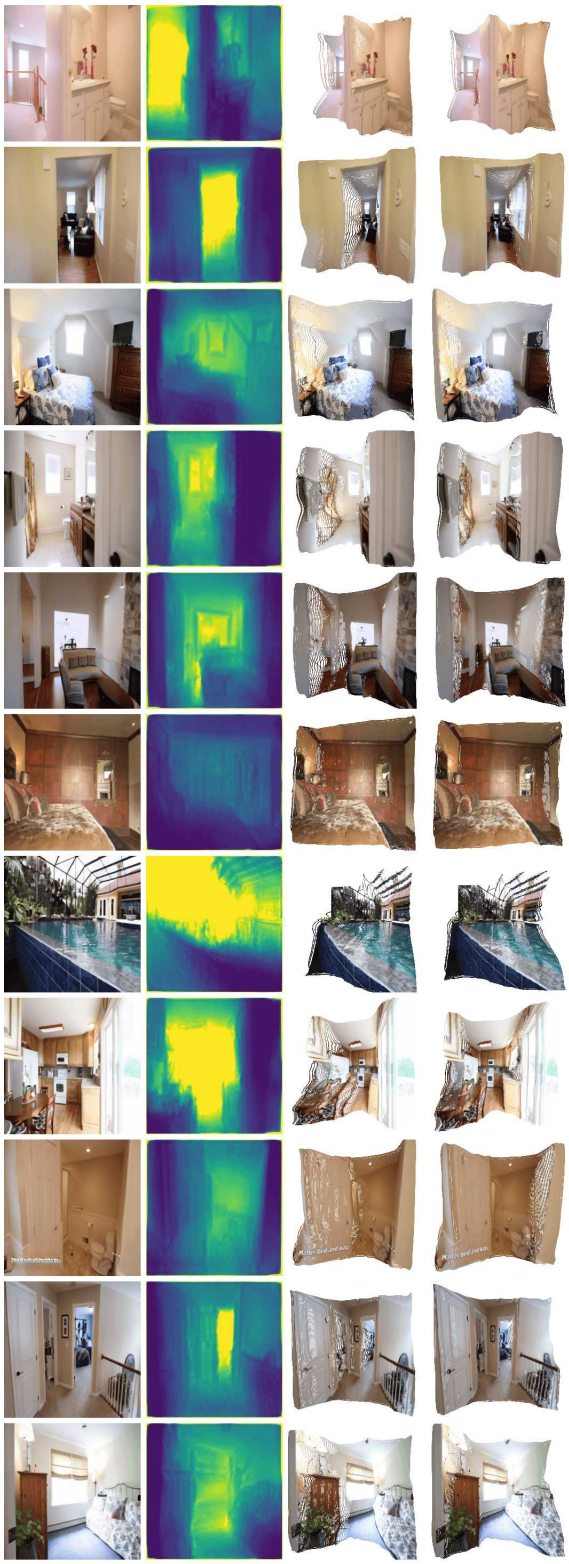


Figure 14: Additional depth predictions on RealEstate10K [74]. We also visualise the point cloud (PC) and the rotated point cloud at $0°$. (Note that the point cloud in the model is actually a point cloud of features, not RGB values.)

B. Additional architectural details

Here we give more information about the precise architectural details used to build the components of our model.

ResNet blocks. Our spatial feature network and refinement networks are composed of ResNet blocks. The ResNet blocks used are the same as those used in [3] (Appendix B, Fig 15 (b)), reproduced in Fig. 15. However, we consider three different setups. The block may be used to increase the resolution of the features using an upsample layer (as used in the original paper by [3]) (Fig. 15(a)). The block may be used to decrease the resolution of the features using an average pooling layer as opposed to the upsample layer (Fig. 15(b)). The block may be used to maintain the resolution of the features using an identity layer as opposed to the upsample layer (Fig. 15(c)).

Spatial feature network. ResNet blocks are stacked together to form the embedding network. In particular, we use the setup in Fig. 16(a).

Refinement network. ResNet blocks are stacked together to form the decoder network. In particular, we use the setup in Fig. 16(b).

Depth regressor. The depth regressor network uses a UNet architecture, as illustrated in Fig. 17.

Additional details on the perceptual loss. We follow the perceptual loss used in [44].

C. Additional details on baselines

In this section, we give further information about the baselines used.

Im to im. We follow the architecture of [75]. However, [75] only considers discrete rotations about the azimuth and a small set of changes in elevation, so [75] takes four values as input, the $\cos$ and $\sin$ values of the azimuth and elevation. However, our datasets include rotation in all three directions, as well as translational motion. As a result, we modify their angle encoder to take 12 values (as opposed to four), and pass the change in viewpoint, `T` to the angle encoder. The network is visualised in Fig. 18.

Vox w/ unet. This baseline is based on [54], which represents 3D shape in a neural network using a voxel representation. Note that they train one model per instance, so their model only generalises to that one object. Their overall setup is as follows. An image is passed through an encoder (*e.g.* our spatial feature network) to obtain a set of features. The features are projected into a voxel grid, which is transformed and projected into the new view. The features are accumulated using an occlusion network, which acts as a pseudo depth predictor and predicts the occupancy of the voxels. The predicted occupancy is used to re-weight and combine features. This is then passed to the decoder (*e.g.* our refinement network) which predicts the scene at the new view. Finally, the generated image is compared to the true image using discriminators and photometric losses.

To reimplement this approach, we follow their architectural choices and use a UNet style architecture for all network components (the spatial feature network, refinement network, and occlusion network). However, we use the discriminators and photometric losses used to train SynSin to ensure that both methods are fair in terms of the discriminator. The details for the encoder/decoder setup are given in Fig. 19. The occupancy network is a 3D UNet, which takes as input the rotated voxels and then predicts occupancy for each voxel location; these are then normalised using a softmax layer over the depth dimension. The details are given in Fig. 20. We use their setup but train the network to generate new images of a scene given a *single* image of a scene.

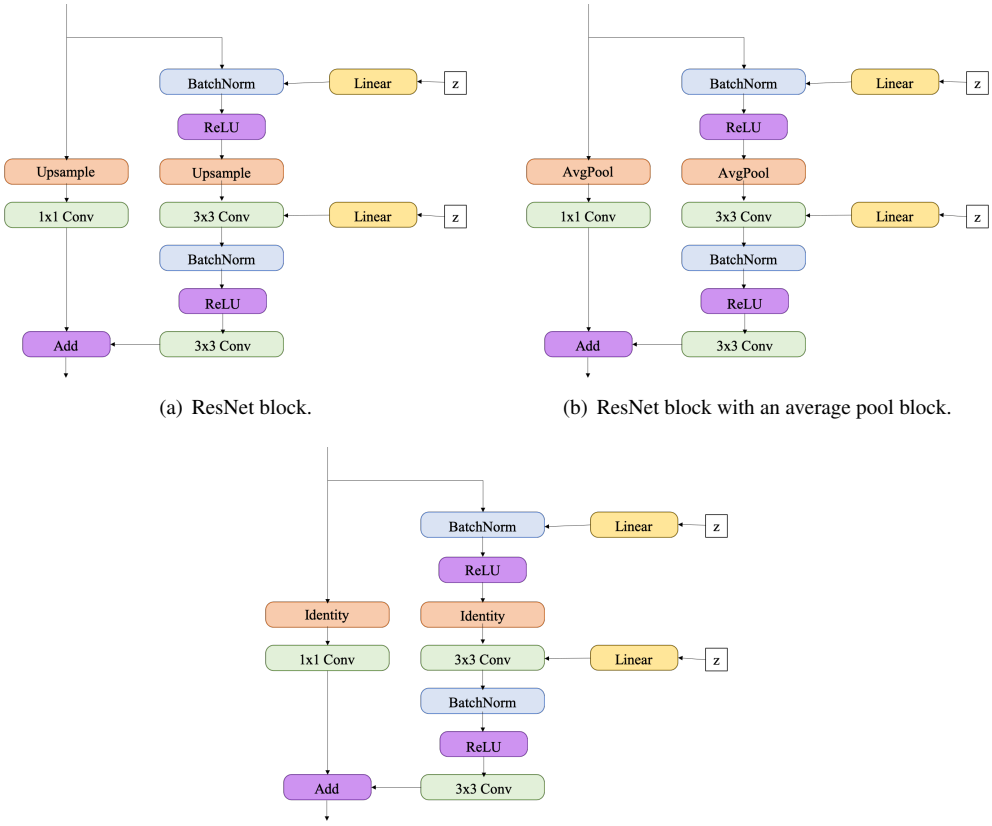
Vox w/ ours. Instead of using the UNet style spatial feature and refinement network in vox w/ ours, we use a sequence of ResNet blocks, as described in Fig. 21. The set of ResNet blocks in the spatial feature network downsamples the image to the appropriate size. The refinement network similarly upsamples the projected features to the appropriate image size. We also use a larger capacity in this setup to ensure that our 3D representation is preferable. The network was trained with a lower learning rate (lr=0.0004) as opposed to (lr=0.001) as in our model, as we found that the model struggled to learn with the higher learning rate.

Other setups. We experimented with other ResNet block sequences and multiple learning rates when creating this baseline. Instead of downsampling the features within the encoder (*e.g.* the spatial feature network), we can use the same spatial feature network as SynSin (to obtain features of size $C \times 256 \times 256$ and then downsample to obtain features of size $C \times 64 \times 64$. Similarly, instead of upsampling the features within the decoder (*e.g.* the refinement network), we can upsample the transformed features to obtain ones of size $C \times 256 \times 256$ and pass these upsampled features to the refinement network and so use the same refinement network we use in SynSin. We found that the results were similar to those of the model used in the paper on RealEstate10K but worse on Matterport.

We additionally found that the results were highly dependent on the learning rate for this model.

3DView. This baseline is based on a depth predictor (*e.g.* [33]), so 3DView predicts depth up to a scale ambiguity.

As the depth is only predicted up to a scale, we generate images for multiple possible scales for each test image and then report results for the best image.

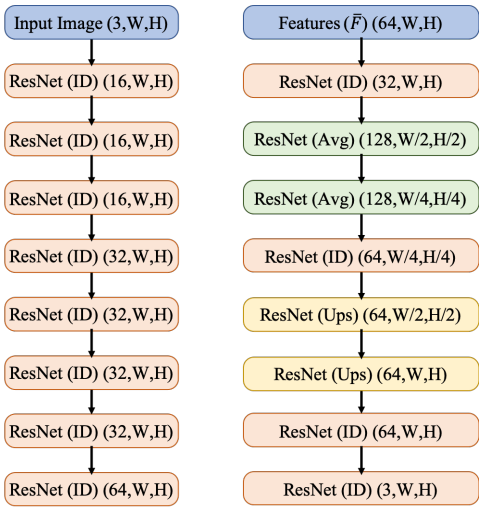


(a) ResNet block.

(b) ResNet block with an average pool block.

(c) ResNet block with an identity block.

Figure 15: An overview of ResNet blocks. In (a), we show the basic ResNet block, (b) when we replace the upsample block by an average pool block, and (c) when we replace the upsample block by an identity block.



(a) Spatial feature network.

(b) Refinement network.

Figure 16: Our sequence of ResNet blocks in the spatial feature and refinement networks.

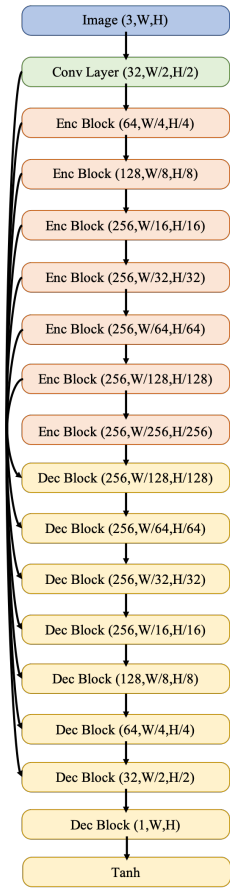


Figure 17: Depth regressor network. An *Enc Block* consists of a sequence of Leaky ReLU, convolution (stride 2, padding 1, kernel size 4), and batch normalisation layers. A *Dec Block* consists of a sequence of ReLU, 2x bilinear upsampling, convolution (stride 1, padding 1, kernel size 3), and batch normalisation layers (except for the final layer, which has no batch normalisation layer).

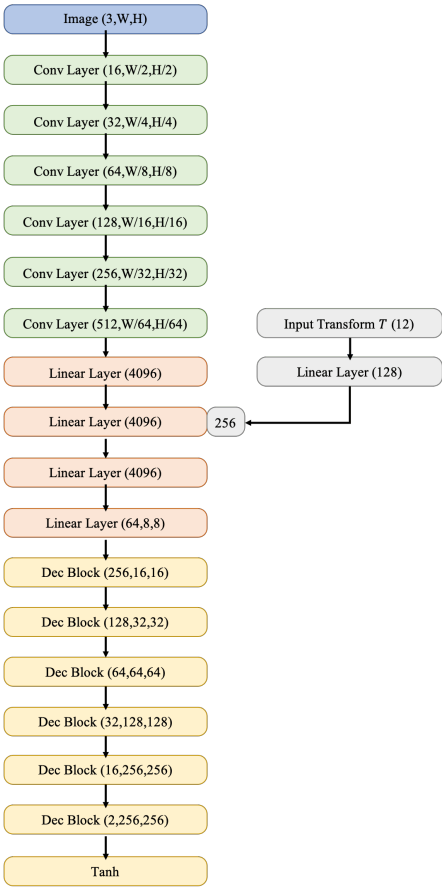
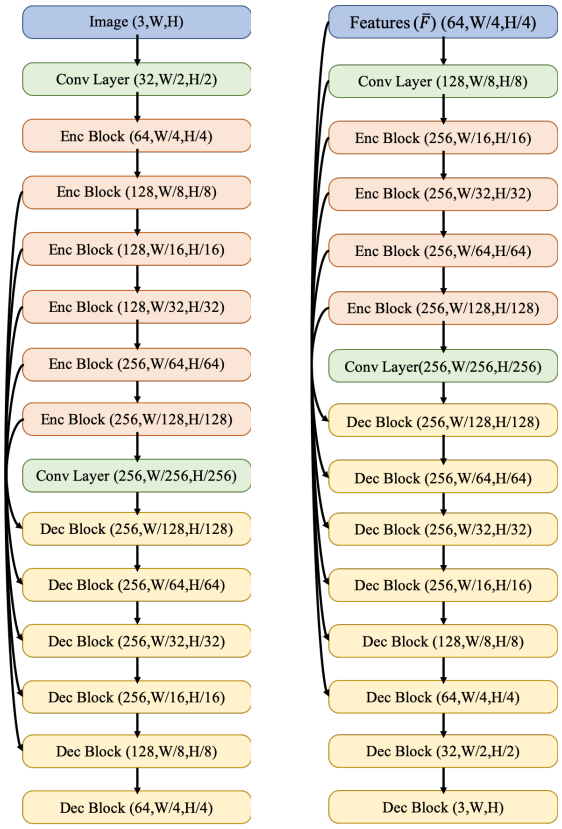


Figure 18: An overview of the image to image network. A *Conv Layer* consists of a sequence of a convolutional layer (stride 2, padding 1, filter size 3), ReLU, and batch normalisation layer. A *Linear Layer* consists of a sequence of a linear layer, ReLU, and batch normalisation layer. A *Dec block* consists of a sequence of a convolutional layer (stride 1, padding 1, filter size 3), ReLU, batch normalisation layer and upsample layer (except for the last, which consists of simply a convolutional layer).



(a) Encoder network. (b) Decoder network.

Figure 19: The encoder and decoder network for the UNet style encoder/decoder setup. An *Enc block* is a sequence of a LeakyReLU, convolutional layer (stride 2, padding 1, kernel size 4) and batch normalisation layer. A *Dec block* is a sequence of ReLU, bilinear upsampling layer, convolutional layer (stride 1, padding 1, kernel size 3), and batch normalisation layer (except for the last layer which has no batch normalisation).

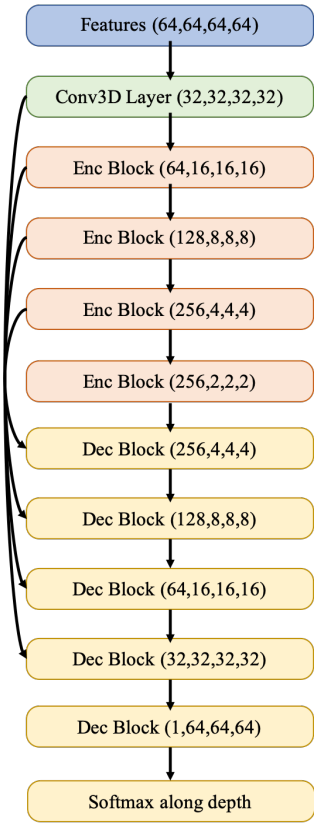
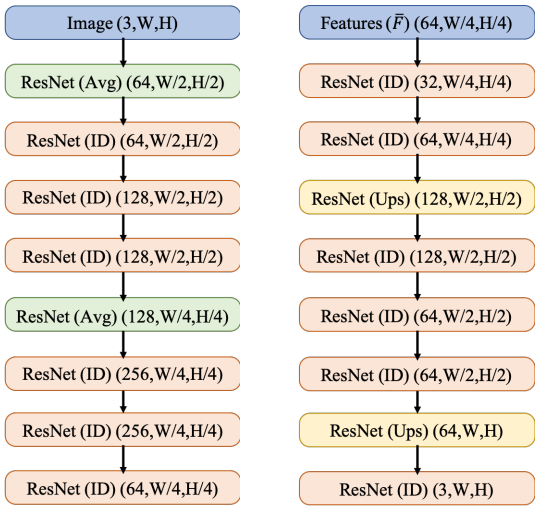


Figure 20: The 3D UNet for predicting the occupancy of voxels. An *Enc block* consists of a sequence of a LeakyReLU, convolutional layer (stride 2, padding 1, kernel size 4) and batch normalisation layer. A *Dec block* consists of a sequence of ReLU, bilinear upsampling layer, convolutional layer (stride 1, padding 1, kernel size 3), and batch normalisation layer (except for the last layer which has no batch normalisation).



(a) Encoder network. (b) Decoder network.

Figure 21: The spatial feature and refinement networks for the ResNet style setup in the Vox w/ ours baseline.

D. Additional information about datasets

Matterport3D. For Matterport, the minimum depth is 0.1 and the maximum depth 10.

RealEstate10K. For RealEstate10K, the minimum depth is 1 and the maximum depth is 100.

KITTI. For KITTI, the minimum depth is 1 and the maximum depth is 50.

E. A description of other setups we tried

Model setup

- We experimented with using a UNet architecture instead of a sequence of ResNet blocks for the spatial feature network and refinement network. This led to much worse results and was more challenging to train.

Differentiable renderer setup

- Other settings for the differentiable renderer: We tried a larger radius, $r = 8$, but this both takes longer to train and gives worse results.
- Other settings for the accumulation function: We tried using a weighted sum with and without normalisation for the accumulation step. These led to similar results, but without normalisation had noisier training characteristics. The implementation of these different accumulation setups is available in the online code.