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 [4]. We evaluated our model on the KITTI dataset in order to compare with [3]. We trained our model on the KITTI dataset and compared with their pretrained model on a held-out set in Table 1. We achieve similar or better results on all metrics. Additionally, because [3] resamples the input image, it cannot generate pixels unseen in the original view. As shown in Fig. 1 (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. 2-3), Replica (Fig. 4), and Matterport3D (Fig. 5). 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. 6-7.

Comparison on KITTI [4]			
	PSNR ↑	SSIM ↑	Perc Sim ↓
SynSin	16.70	0.52	**2.07**
ContView [3]	**16.90**	**0.54**	2.21

Table 1: Comparison on KITTI to [3]. ↑ denotes higher is better, ↓ lower is better.

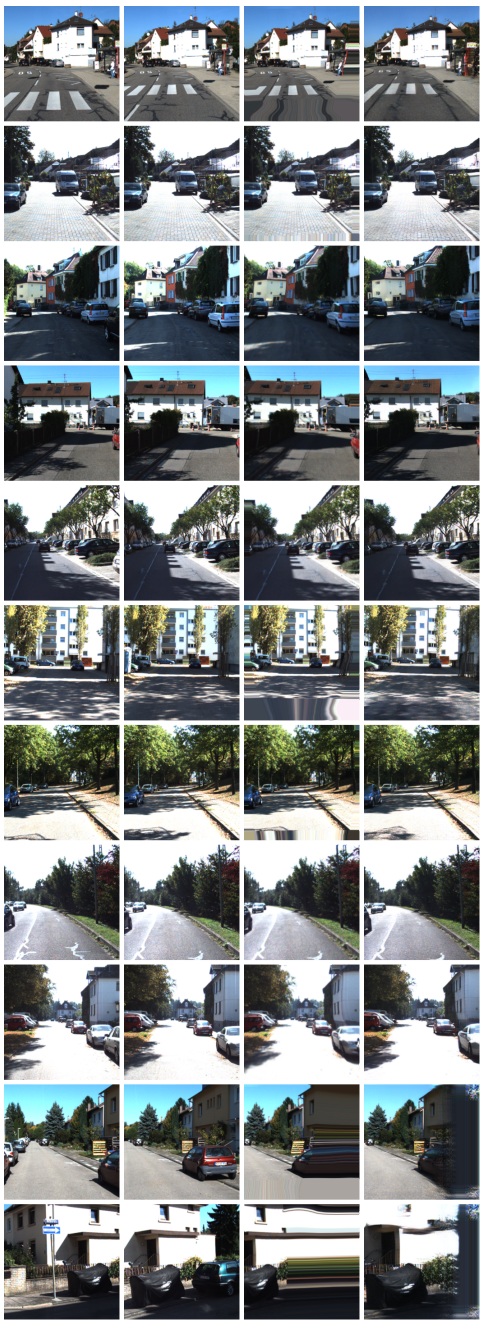


Figure 1: Qualitative results on KITTI [9] comparing SynSin to [3]. The bottom two rows demonstrate failure cases due to large viewpoint change. Zoom in for details.

Figure 2: Additional results on RealEstate10K [9]. Zoom in for details.

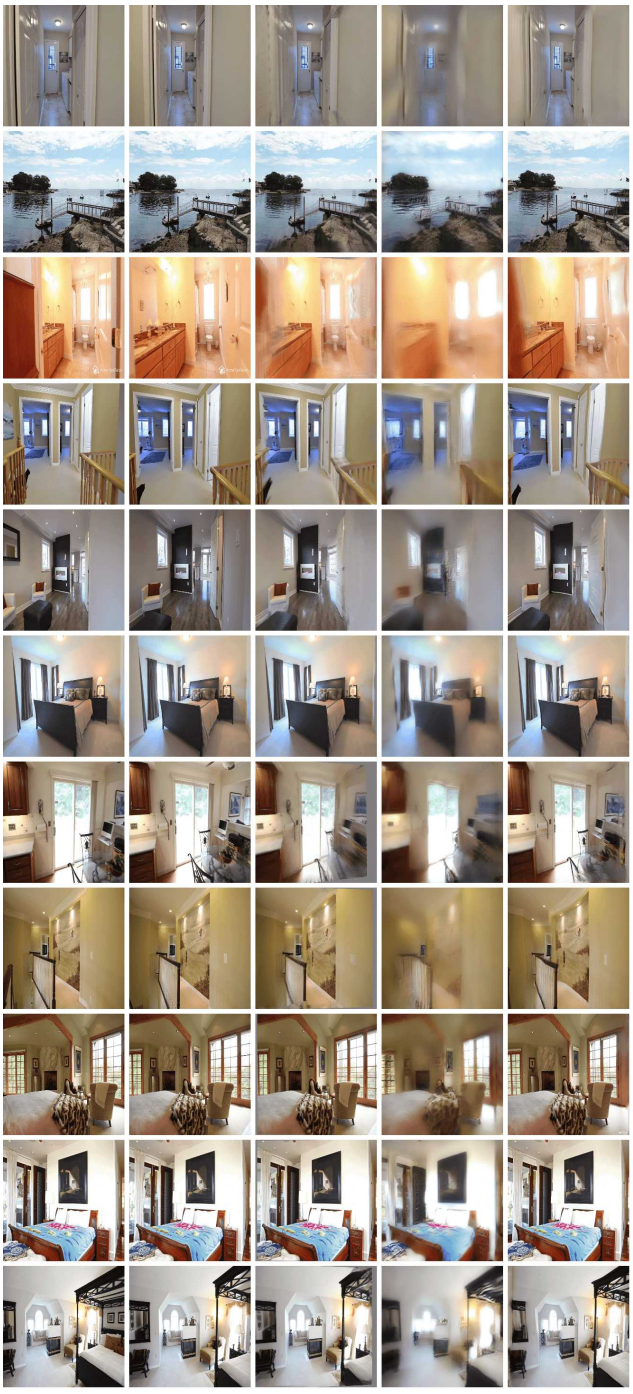


Figure 3: Additional results on RealEstate10K [9]. Zoom in for details.

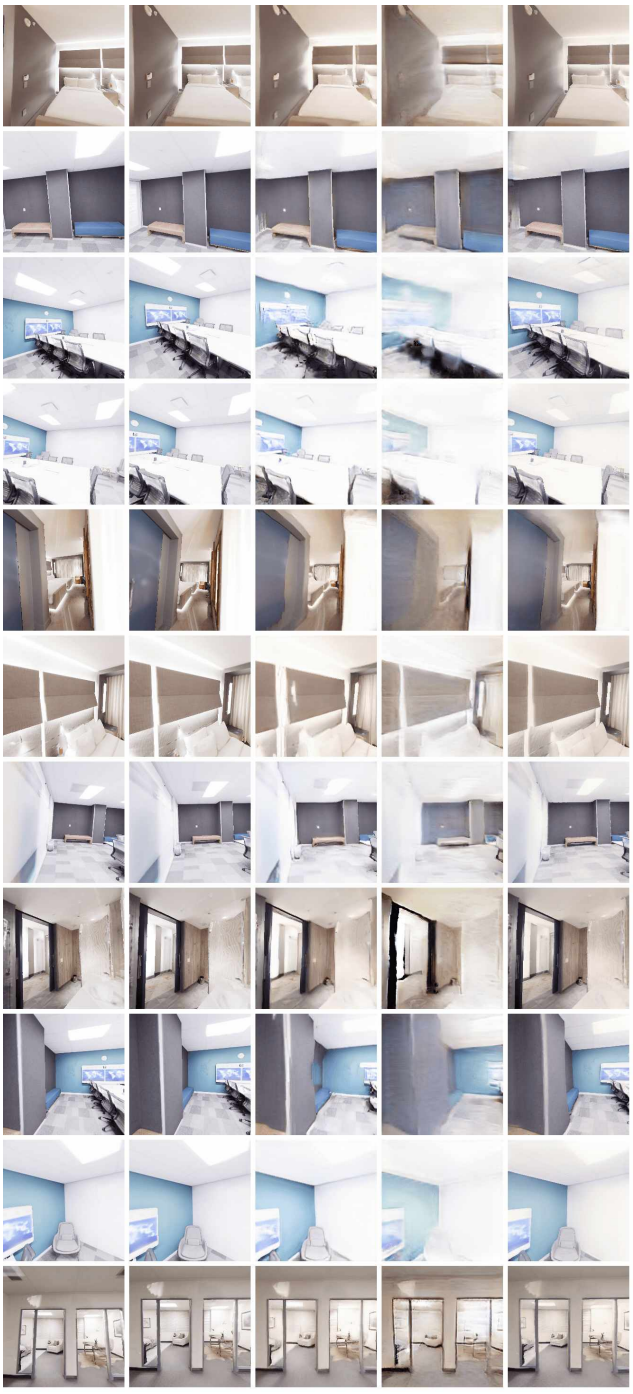


Figure 4: Additional results on Replica [8]. Zoom in for details.

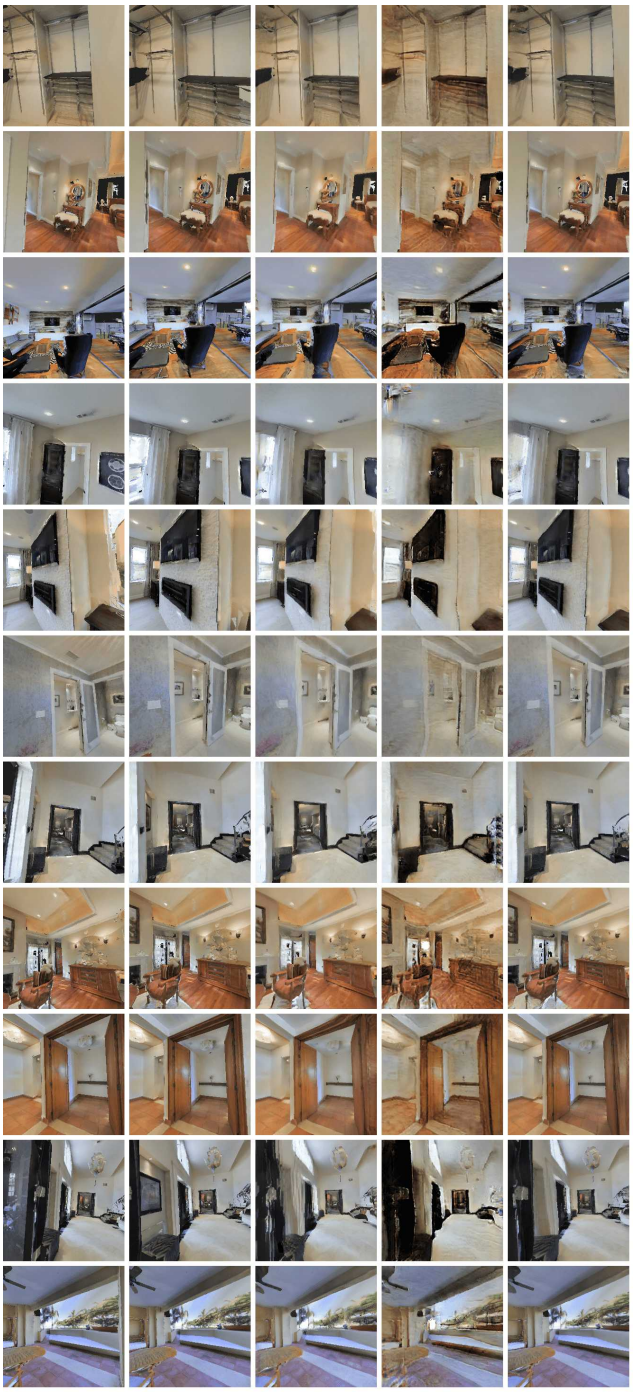


Figure 5: Additional results on Matterport3D [2]. Zoom in for details.

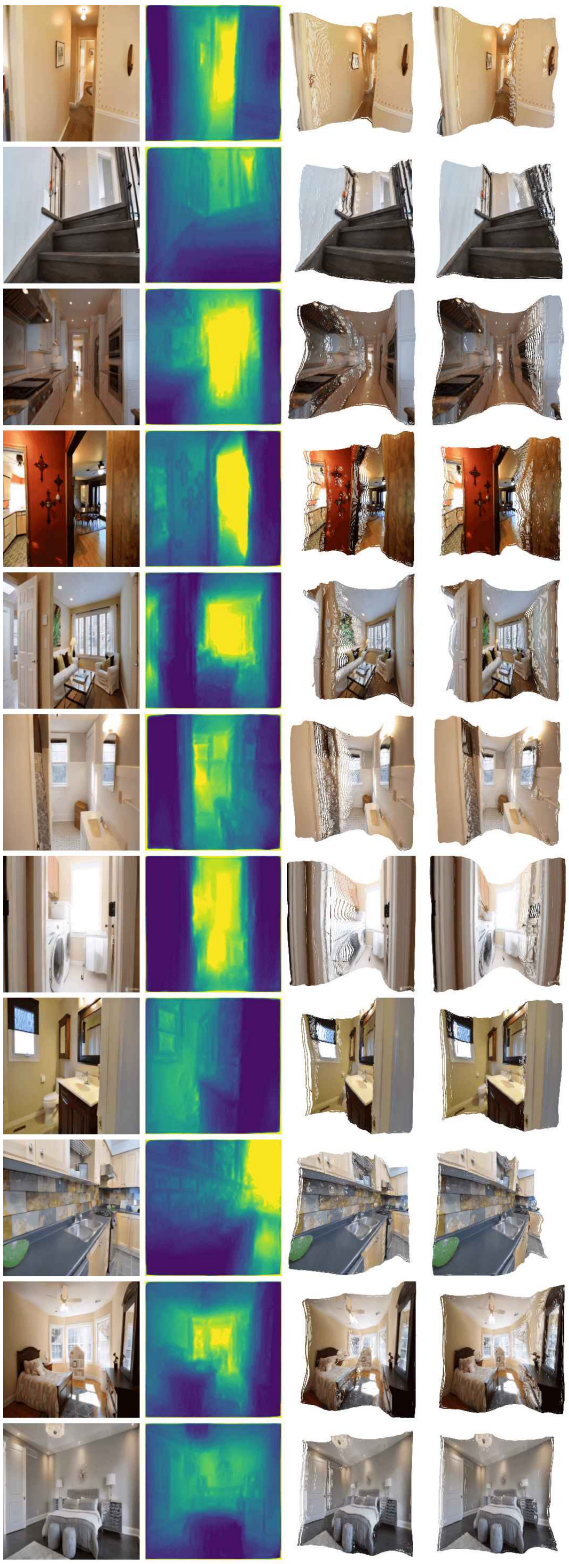


Figure 6: Additional depth predictions on RealEstate10K [9]. We also visualise the point cloud (PC) and the rotated point cloud at $-45°$. (Note that the point cloud in the model is actually a point cloud of features, not RGB values.)

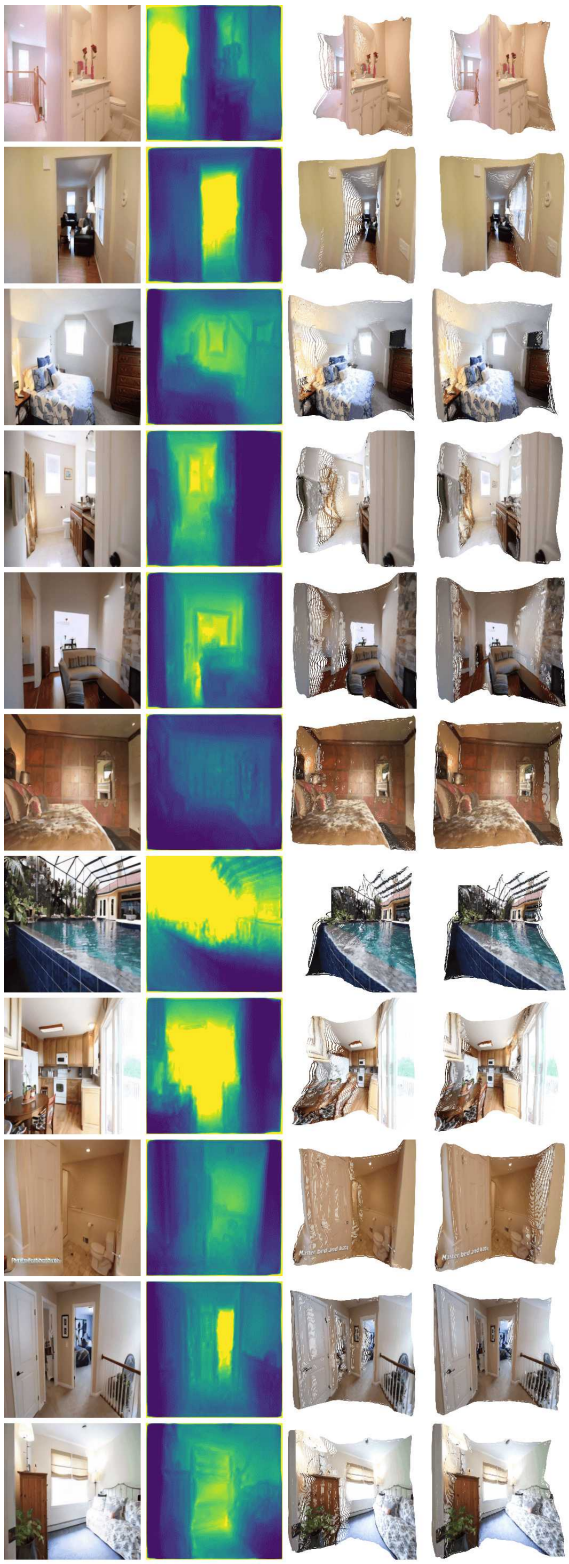


Figure 7: Additional depth predictions on RealEstate10K [9]. We also visualise the point cloud (PC) and the rotated point cloud at $0°$. (Note that the point cloud in the model is actually a point cloud of features, not RGB values.)

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 [1] (Appendix B, Fig 15 (b)), reproduced in Fig. 8. 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 [1]) (Fig. 8(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. 8(b)). The block may be used to maintain the resolution of the features using an identity layer as opposed to the upsample layer (Fig. 8(c)).

Spatial feature network. ResNet blocks are stacked together to form the embedding network. In particular, we use the setup in Fig. 9(a).

Refinement network. ResNet blocks are stacked together to form the decoder network. In particular, we use the setup in Fig. 9(b).

Depth regressor. The depth regressor network uses a UNet architecture, as illustrated in Fig. 10.

Additional details on the perceptual loss. We follow the perceptual loss used in [6].

C. Additional details on baselines

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

Im to im. We follow the architecture of [10]. However, [10] only considers discrete rotations about the azimuth and a small set of changes in elevation, so [10] 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. 11.

Vox w/ unet. This baseline is based on [7], 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. 12. 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. 13. We use their setup but train the network to generate new images of a scene given a *single* image of a scene.

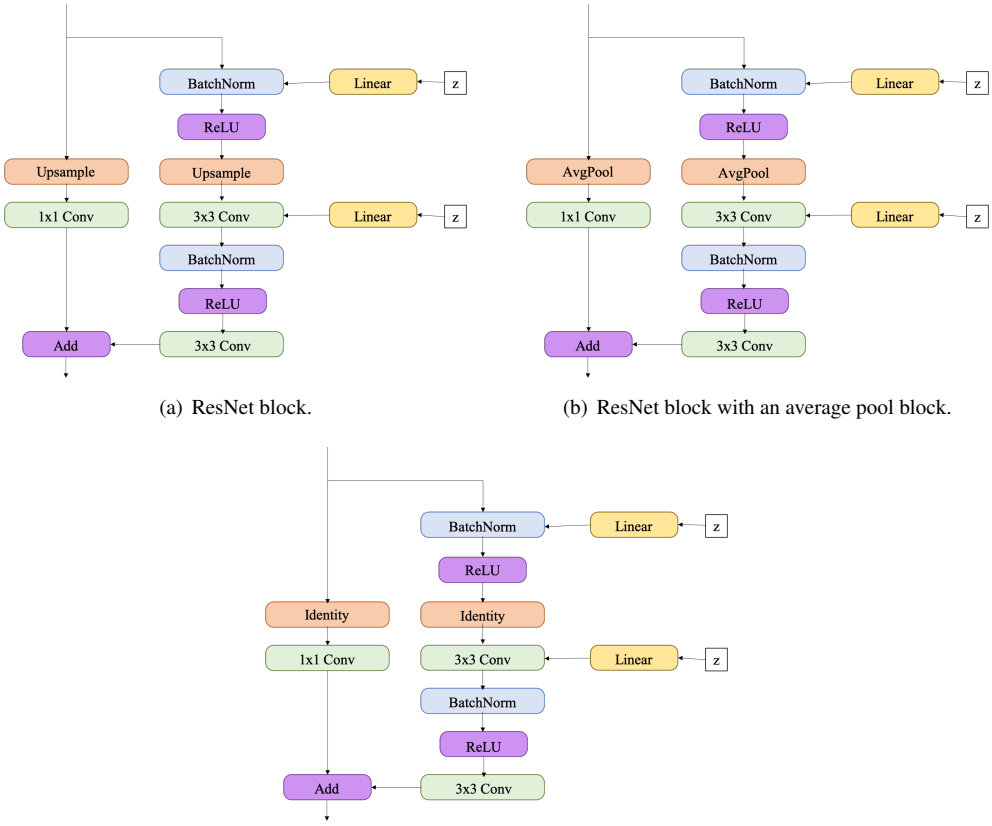
Vox w/ ours. Instead of using the UNet style spatial feature and refinement network in vox w/ ours, we use a sequence of ResNet blocks, as described in Fig. 14. The set of ResNet blocks in the spatial feature network downsamples the image to the appropriate size. The refinement network similarly upsamples 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.* [5]), 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.

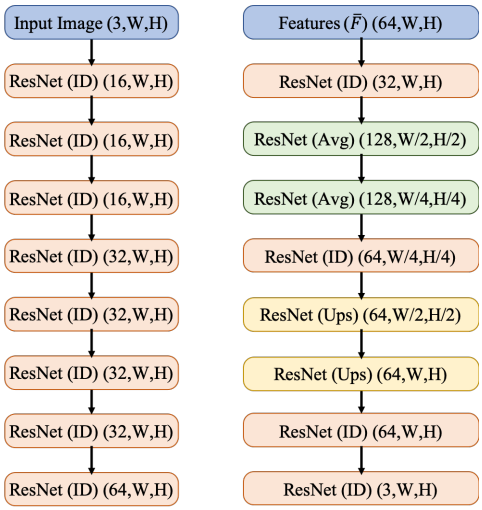


(a) ResNet block.

(b) ResNet block with an average pool block.

(c) ResNet block with an identity block.

Figure 8: 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 9: Our sequence of ResNet blocks in the spatial feature and refinement networks.

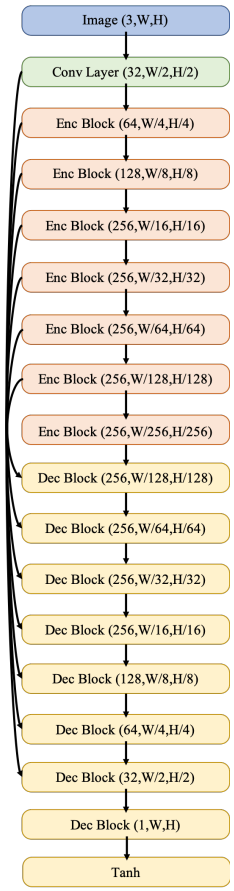


Figure 10: Depth regressor network. An *Enc Block* consists of a sequence of Leaky ReLU, convolution (stride 2, padding 1, kernel size 4), and batch normalisation layers. A *Dec Block* consists of a sequence of ReLU, 2x bilinear upsampling, convolution (stride 1, padding 1, kernel size 3), and batch normalisation layers (except for the final layer, which has no batch normalisation layer).

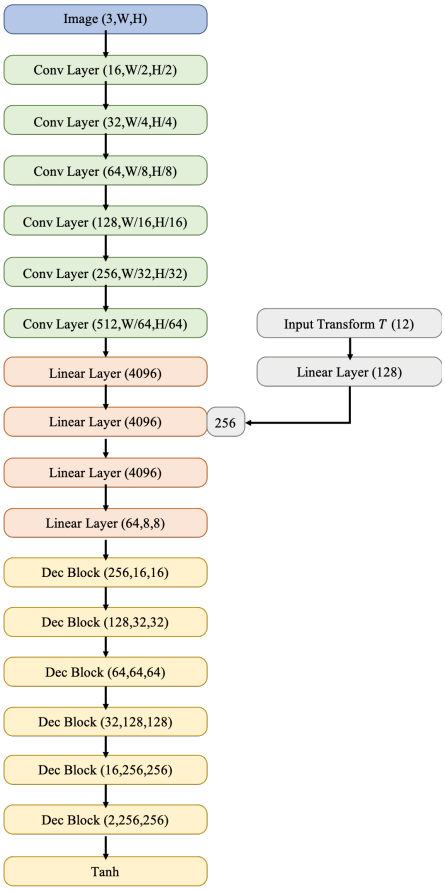
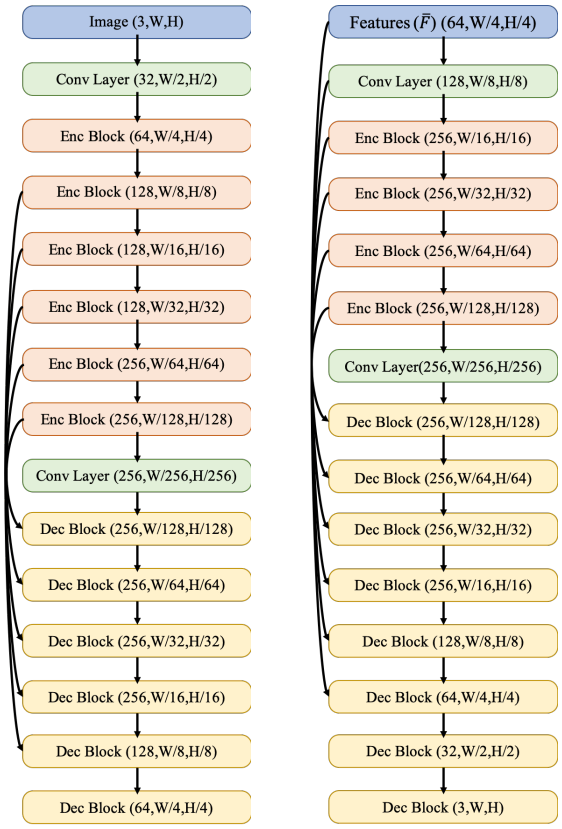


Figure 11: An overview of the image to image network. A *Conv Layer* consists of a sequence of a convolutional layer (stride 2, padding 1, filter size 3), ReLU, and batch normalisation layer. A *Linear Layer* consists of a sequence of a linear layer, ReLU, and batch normalisation layer. A *Dec block* 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 12: 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).

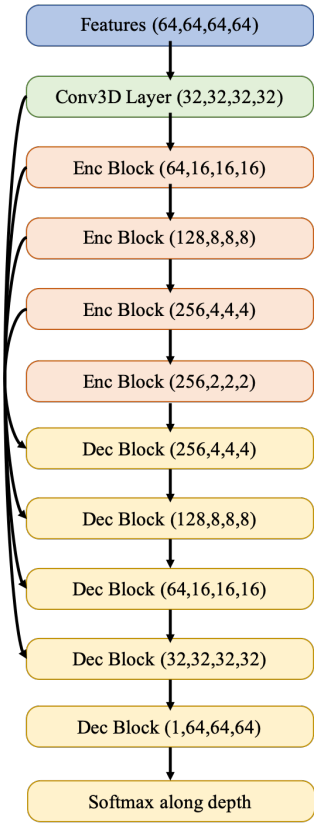
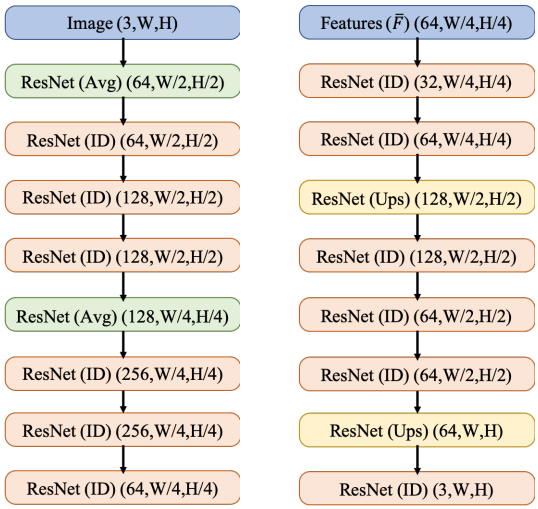


Figure 13: The 3D UNet for predicting the occupancy of voxels. An *Enc block* consists of a sequence of a LeakyReLU, convolutional layer (stride 2, padding 1, kernel size 4) and batch normalisation layer. A *Dec block* consists of a sequence of ReLU, bilinear upsampling layer, convolutional layer (stride 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 14: 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.

References

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

[2] 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/`.

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

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

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

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

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

[8] 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.

[9] 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.

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