Worldsheet: Wrapping the World in a 3D Sheet for View Synthesis from a Single Image

Ronghang Hu[1] Nikhila Ravi[1] Alexander C. Berg[1] Deepak Pathak[2]
[1]Facebook AI Research (FAIR) [2]Carnegie Mellon University

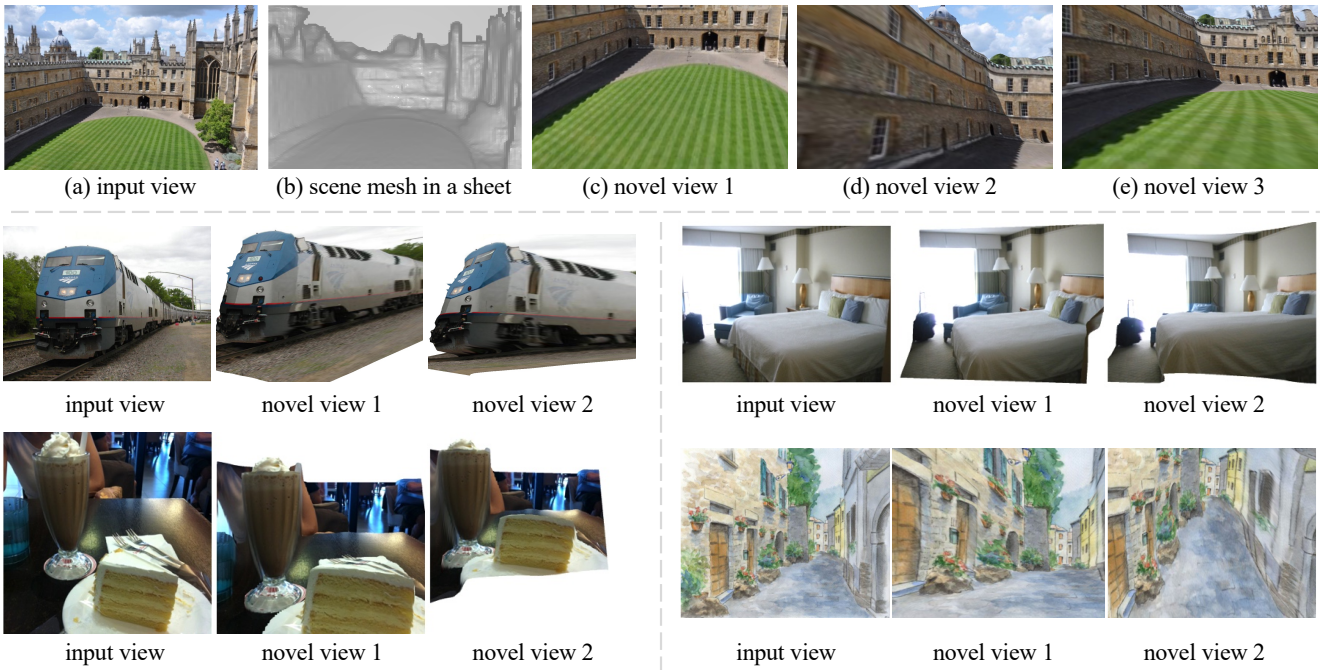


Figure 1: We synthesize novel views from large viewpoint changes given a single input RGB image (shown in a) by wrapping a mesh sheet (shown in b) onto the image, and rendering it from novel viewpoints (shown in c, d, e). Plausible novel views are generated for outdoor scenes, outdoor objects, indoor scenes, indoor objects and even paintings with high resolution (960×960) input. Please see continuously synthesized views at `worldsheet.github.io` (Image sources: [49, 13, 25, 28]).

Abstract

We present Worldsheet, a method for novel view synthesis using just a single RGB image as input. The main insight is that simply shrink-wrapping a planar mesh sheet onto the input image, consistent with the learned intermediate depth, captures underlying geometry sufficient to generate photorealistic unseen views with large viewpoint changes. To operationalize this, we propose a novel differentiable texture sampler that allows our wrapped mesh sheet to be textured and rendered differentiably into an image from a target viewpoint. Our approach is category-agnostic, end-to-end trainable without using any 3D supervision, and requires a single image at test time. We also explore a simple extension by stacking multiple layers of Worldsheets to better handle occlusions. Worldsheet consistently outperforms prior state-of-the-art methods on single-image view synthesis across several datasets. Furthermore, this simple idea captures novel views surprisingly well on a wide range of high-resolution in-the-wild images, converting them into navigable 3D pop-ups. Video results and code are available at https://worldsheet.github.io.

1. Introduction

A 2D image is the projection of an underlying 3D world, but as humans, we have no trouble in understanding this structure and imagining how an image will look from other views. Consider the train shown in Figure 1, we can seamlessly predict other views from a single image based on the abstractions we have learned from past experience of seeing several trains, or similar shaped objects from different views. Enabling machines with such an ability to reason about 3D from a single image will bring trillions of still photos to life, with wide applications in virtual reality, animation, image editing, and robotics.

The goal of synthesizing novel views from 2D images has been pursued for decades, from early efforts relying completely on multi-view geometry [7, 61, 38], to more recent learning based approaches [59, 43, 45, 1, 9, 30, 26, 35, 23, 51, 54]. Over the years, significant progress has been made in this direction. However, despite impressive photorealistic output renderings, most of these previous approaches require multiple images or ground-truth depth at test time, which severely hinders their practicality. To compensate for the lack of multiple views or 3D models at test time, methods for single-image 3D rely on statistical learning from data. This line of work can be traced back to classic works of Hoiem *et al*. [13], followed by Saxena *et al*. [37], that obtain 'qualitative 3D' from a single image by fitting a collection of planes onto the image.

An ideal approach to general-purpose view synthesis should not only rely on a single image at test time, but also learn from easy-to-collect supervision signal during training. In the deep learning era, there is growing interest in end-to-end methods with intermediate 3D representations supervised by multiple images and no explicit 3D information during training. However, they are mostly applied to objects [20, 47, 55, 15, 43], and are either category-specific, restricted to synthetic scenes, or both. Recent works [5, 53, 48] address these issues by training with multiple views of real-world scenes, relying on point cloud or multiplane images as intermediate representations. However, multiplane images only perform well with relatively small viewpoint changes as each plane is at a constant depth; for point clouds, one needs to represent each point in a scene individually, making it inefficient to scale to high-resolution data or large viewpoint changes. In contrast, meshes can provide a sparser scene representation, *e.g*., two triangular mesh faces can theoretically represent the entire flat surface of a wall, making it ideal for single-image view synthesis. However, mesh recovery from single images has been studied mostly for object images and in a category-specific manner [2, 15, 19] and not for scenes.

In this paper, we present an end-to-end approach for novel view synthesis from *a single image* of a scene via an intermediate *mesh representation*. Unlike mesh reconstruction for objects of specific categories, generating meshes for a scene is challenging as there is no notion of mean or canonical shape to start from, or silhouette from segmentation for supervision. We circumvent this problem by wrapping a deformable mesh sheet over the 3D world – much like wrapping a 2D tinfoil onto a 3D pan before baking! We name this shrink-wrapped mesh *Worldsheet*, a term borrowed from physics for the 2D manifold of high-dimensional strings. After generating this Worldsheet for a given view, novel views are obtained by moving the camera in 3D space (Figure 2), which allows us to train from just two views of a scene using only rendering losses *without any 3D or depth supervision*.

To train our model end-to-end, both reconstruction of the mesh texture from input view and rendering from a novel camera view need to be differentiable. The latter is easily handled thanks to recent differentiable mesh renderers [16, 27, 33]. To address the former, we propose a differentiable texture sampler over projected 2D views, enabling gradient computation of the reconstructed texture map over the 3D mesh geometry. Furthermore, to better handle occlusions and depth discontinuities, we propose a simple extension by stacking multiple layers of Worldsheets onto the scene.

In summary, Worldsheet generates novel views by learning to predict scene geometry from a single image. Although 3D mesh reconstruction via differentiable rendering is common for objects, to our best knowledge, this is the first work to show mesh recovery for *scenes* just from multi-view supervision. Our model consistently outperforms prior state-of-the-art by a significant margin on three benchmark datasets (Matterport [3], Replica [46], and RealEstate10K [59]), and is applicable to very high-resolution images in-the-wild as shown in Figure 1.

2. Related work

Novel view synthesis from multiple images. Traditional novel view synthesis methods use multiple input views at test time [4, 22, 11], and are often based on different representations. Among recent works, Waechter *et al*. [50] build scene meshes with diffuse appearance. StereoMag [59] proposes multiplane images (MPIs) from a stereo image pair as a layered scene representation. NPBG [1] captures the scene as a point cloud with neural descriptors. NeRF [30] proposes a neural radiance field representation for scene appearance, and is followed by many extensions (see [8] for a summary). NSVF [26] adopts sparse voxel octrees as scene representations. FVS [34] and SVS [35] blend multiple source images based on a geometric scaffold. Yoon *et al*. [56] combine depth from both single and multiple views to generate novel views of dynamic scenes. Access to multiple input views greatly simplifies the task, allowing the scene geometry to be recovered via multi-view stereo [38].

Novel view synthesis from a single image. In early works, Debevec *et al*. [7] recover 3D scene models and Horry *et al*. [14] fit a regular mesh to generate novel views. Liebowitz *et al*. [24] and Criminsi *et al*. [6] generate meshes via projective geometry constraints but these methods came at the expense of manual editing. Hoiem *et al*. [13] generate automatic 3D pop-up by fitting vertical and ground planes onto the 2D image, unlike our mesh representation. More recently in [31, 18, 40], layered depth images are used for single image view synthesis based on a pre-trained depth estimator. In [48], online videos are used to train a scale-invariant MPI representation for view synthe-

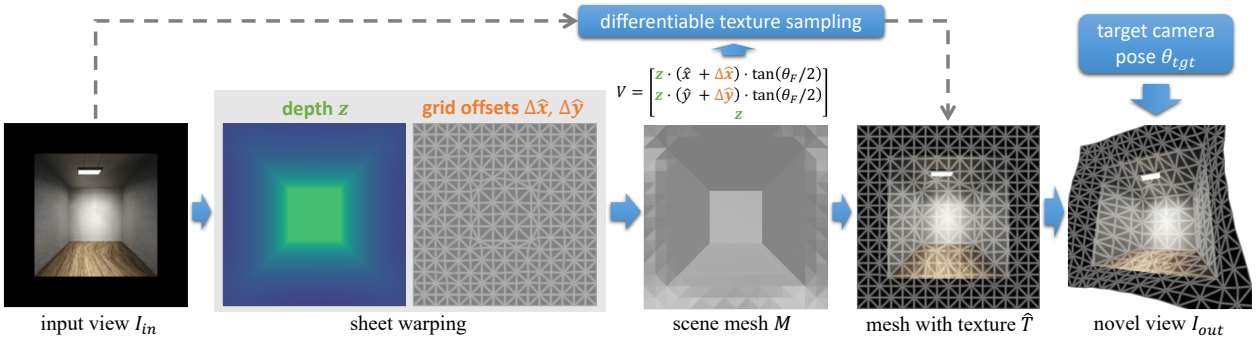


Figure 2: An overview of our *Worldsheet* approach. Given an input view I_{in}, we build a scene mesh by warping a $W_m \times H_m$ grid sheet onto the scene geometry via grid offset $(\Delta\hat{x}, \Delta\hat{y})$ and depth z (Sec. 3.1). Then, we sample the UV texture map $\hat{T}$ of the scene mesh differentiably (Sec. 3.2) and render it from the target camera pose to output a novel view I_{out}. Our mesh warping is learned end-to-end using the losses on the novel view (Sec. 3.3). We further apply an inpainting network over I_{out} (not shown above; see Sec. 3.4) to inpaint invisible regions and refine image details, outputting a refined novel view I_{paint}. We train with two views without any 3D or depth supervision and require just a single RGB image at test time.

sis. SynSin [53] synthesizes novel views from a single image with a feature point cloud. In contrast, we learn to construct scene meshes instead of point clouds and directly map image texture instead of feature vectors to generate novel views from large viewpoint changes.

Differentiable mesh rendering. Recent work on differentiable mesh renders [16, 27, 33] allow learning 3D structures through synthesis. NMR [16] and SoftRas [27] reconstruct the 3D object shape as a mesh by rendering it, comparing it with the input image, and back-propagating losses to refine the mesh geometry. CMR [15], CSM [19] and U-CMR [10] build category-specific object meshes from images by deforming from a mean or template category shape through silhouette (and keypoints in [15]) supervision.

Our method is aligned with the analysis-by-synthesis paradigm above. However, unlike most previous works that apply differentiable mesh rendering to objects, we learn the 3D geometry of *scenes* through the rendering losses on the novel view. Moreover, instead of predicting a texture flow as in [15, 10], we propose to analytically sample the mesh texture from the input view with a differentiable texture sampler. Unlike [32], our differentiable texture sampler considers multiple mesh faces in the z-buffer (soft rasterization instead of only the closest one), and assumes perspective (instead of orthographic) camera projection.

3. Worldsheet: Rendering the World in a Sheet

In this work, we propose *Worldsheet* to synthesize novel views from a single image, as shown in Figure 2. Our model build a 3D scene mesh M by warping a lattice grid (*i.e.* a "sheet") onto the scene geometry, and is trained with only 2D rendering losses *without any 3D or depth supervision*.

3.1. Scene mesh prediction by warping a sheet

From the input view image I_{in} of size $W_{im} \times H_{im}$, we build a scene mesh by warping a $W_m \times H_m$ lattice grid (*i.e.* a sheet) onto the scene, as shown in Figure 2. We first extract a $W_m \times H_m$ visual feature map $\{q_{w,h}\}$ from I_{in} with a convolutional neural network. Each $q_{w,h}$ is a feature vector at spatial location (w, h) on the $W_m \times H_m$ network output. In our implementation, we use ResNet-50 [12] (pretrained on ImageNet) with dilation [57] to output features $\{q_{w,h}\}$.

From each $q_{w,h}$ on the feature map, we predict the grid offset $\Delta\hat{x}_{w,h}$ and $\Delta\hat{y}_{w,h}$ to decide how much the vertex (w, h) on the grid should move away from its anchor positions within the image plane (we output $\Delta\hat{x}_{w,h}$ and $\Delta\hat{y}_{w,h}$ in NDC space [41] between -1 to 1). We also predict how far each vertex is from the camera, *i.e.* its depth $z_{w,h}$. These values are predicted using learned mappings as

$$\Delta\hat{x}_{w,h} = \tanh\left(W_1 q_{w,h} + b_1\right)/(W_m - 1) \quad (1)$$

$$\Delta\hat{y}_{w,h} = \tanh\left(W_2 q_{w,h} + b_2\right)/(H_m - 1) \quad (2)$$

$$z_{w,h} = g\left(W_3 q_{w,h} + b_3\right) \quad (3)$$

where division by $(W_m - 1)$ and $(H_m - 1)$ ensures that the vertices can only move within a certain range. $g(\cdot)$ is a scalar nonlinear function to scale the network prediction into depth values. We use $g(\psi) = \alpha_g/(\sigma(\psi) + \epsilon_g) + \beta_g$ in our implementation, where $\sigma(\cdot)$ is the sigmoid function and α_g, β_g and ϵ_g are fixed hyper-parameters.

Building the 3D scene mesh. We first build the mesh vertices $\{V_{w,h}\}$ from the grid offset and depth as

$$V_{w,h} = \begin{bmatrix} z_{w,h} \cdot (\hat{x}_{w,h} + \Delta\hat{x}_{w,h}) \cdot \tan(\theta_F/2) \\ z_{w,h} \cdot (\hat{y}_{w,h} + \Delta\hat{y}_{w,h}) \cdot \tan(\theta_F/2) \\ z_{w,h} \end{bmatrix} \quad (4)$$

for $w = 1, \cdots, W_m$ and $h = 1, \cdots, H_m$. Here θ_F is the camera field-of-view, and $\hat{x}_{w,h}$ and $\hat{y}_{w,h}$ are anchor positions on the grid equally spaced from -1 to 1.

Then, we connect the mesh vertices $\{V_{w,h}\}$ along the edges on the grid to form mesh faces $\{F\}$ as shown in Figure 2 and obtain a 3D mesh $M = (\{V_{w,h}\}, \{F\})$. A vertex in the mesh is connected to its 4 or 8 neighbours on the grid.

To encourage the mesh surface to be smooth unless it needs to bend to fit the scene geometry, we apply a Laplacian term $L_m = \sum_{w,h} \left\| \sum_{(\bar{w},\bar{h})\in N(w,h)} \left(V_{\bar{w},\bar{h}} - V_{w,h}\right) \right\|_1$ on the mesh vertices, where $N(w,h)$ are the adjacent vertices to (w,h). In addition, we also apply an L2 regularization term $L_g = \sum_{w,h} \left(\Delta\hat{x}^2_{w,h} + \Delta\hat{y}^2_{w,h}\right)$ to the grid offset.

3.2. Differentiable texture sampler

To render the input scene in another camera pose for novel view synthesis, we need to project image texture from the input view to the target view in a differentiable manner. While existing renderers [16, 27, 33] can render an image from a scene mesh based on its texture map, they cannot directly transform image pixels in screen space *between* two different camera poses. In our model, we accomplish differentiable projection between two views by *first reconstructing the scene mesh's texture map from the input view* (which involves inverting the texture-map-to-image perspective transform in a differentiable manner) so that it can be later rendered with the scene mesh in novel views using existing mesh renderers.

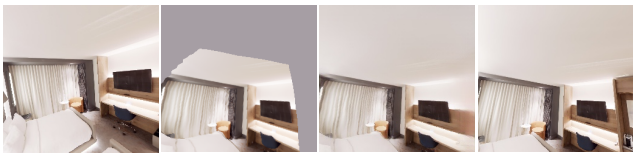
While a few approaches [15, 10] build a mesh texture map with a learned texture flow on objects, it is hard to apply the same to scenes, which do not have canonical shapes. Here, we take an alternative route and propose a differentiable texture sampler. We *analytically* sample the mesh texture $\hat{T}$ as a UV texture map [41] from the input view I_{in}, where gradients $\partial\hat{T}/\partial V$ and $\partial\hat{T}/\partial I_{in}$ over the vertex coordinates and the input image respectively can be computed.

To implement this texture sampler, we project the mesh faces onto the image plane to build a buffer (sorted in ascending z-order) containing the z values and 2D euclidean distance of points on the closest K mesh faces whose projection overlaps image pixel $p_{i,j}$ as in PyTorch3D [33]. Then, we splat the RGB pixel intensities from the image I_{in} onto the UV texture map $\hat{T}$. Specifically, we first compute the weight $w^k_{i,j}$ denoting the contribution of the k-th face color on pixel $p_{i,j}$ based on the softmax blending formulation in [33, 27]. We then decompose the input image I_{in} into K images I^k_{in}, where $I^k_{in}(i,j) = I_{in} \cdot w^k_{i,j}$, and splat the RGB pixels from each I^k_{in} to a texture map layer $\hat{T}^k$ as

$$\hat{T}^k = \text{splat}(I^k_{in}, f^k) \tag{5}$$

where the flow $(u,v) = f^k(i,j)$ maps image coordinates (i,j) to UV coordinates (u,v) on the k-th mesh face in the z-buffer. Here splat is a differentiable splatting operation from the image space I^k_{in} to the texture space $\hat{T}^k$. Finally, we sum all the K texture maps as the final UV texture map $\hat{T} = \sum_k \hat{T}^k$. Please see supplemental for more details.

In summary, the image pixels are splatted onto the texture space via each rasterized mesh face, and blended together to obtain the final texture map. The entire process is differentiable with respect to both I_{in} and the mesh vertex coordinates $\{V\}$, as one can analytically compute $\partial\hat{T}^k/\partial I^k_{in}$, $\partial\hat{T}^k/\partial f^k$, $\partial f^k/\partial V$, and $\partial w^k_{i,j}/\partial V$.



(a) input view 2D image I_{in} (b) mesh output target view I_{out} (c) inpainting on target view I_{paint} (d) ground-truth target view I_{tgt}

Figure 3: Parts of the target view (the grey area in b) are often invisible from the input and must be imagined based on prior knowledge. We make plausible predictions over the invisible regions with an inpainting network (c). However, this task is inherently uncertain (*e.g.* one cannot be sure about the rightmost cabinet in d).

3.3. Learning scene geometry by view synthesis

To synthesize a novel view, we project the mesh vertex coordinates $\{V\}$ from the input camera pose θ_{in} to $\{V^{tgt}\}$ in the camera coordinate space of the target viewpoint θ_{tgt}. Then, we render the mesh $M^{tgt} = (\{V^{tgt}\}, \{F\})$ in the target camera pose along with its texture map $\hat{T}$ to output a 2D image I_{out} of size $W_{im} \times H_{im}$ as the target view:

$$I_{out} = \text{render\_mesh}(\{V^{tgt}\}, \{F\}, \hat{T}). \tag{6}$$

We use the differentiable mesh renderer in [33] so that we can compute the gradients $\partial I_{out}/\partial V^{tgt}$ and $\partial I_{out}/\partial\hat{T}$. Through mesh rendering, we also obtain a foreground mask F_{out} with the same size as I_{out}, indicating which pixels in the rendered image I_{out} are covered by the mesh and which pixels are from background color, as shown by the grey area in Figure 3 (b).

Our model is supervised with paired input and target views of a scene (along with their camera poses). We use a pixel L1 loss $L^{rgb}_{out} = \|I_{out} - I_{tgt}\|_1/(W_{im} \cdot H_{im})$ and a perceptual loss [52, 53] $L^{pc}_{out} = P(I_{out}, I_{tgt})$, where I_{tgt} is the ground-truth target view image. The model then needs just a single image at test time.

3.4. Inpainting and image refinement

The target view image consists of two parts: things that can be directly seen from the input view I_{in}, and things that need to be imagined based on our prior knowledge of the visual world, as illustrated in Figure 3. As our mesh warping and rendering procedure in Sec. 3.1, 3.2 and 3.3 builds a pixel-to-pixel correspondence between the input and the target view, it only renders pixels that are *visible* from the input view. To obtain a plausible imagination of the *invisible* image regions, we apply an inpainting network G on the rendered mesh I_{out} to fill the missing regions and output a new image $I_{paint} = G(I_{out})$ as the final target view.

#	Method	Matterport [3] PSNR ↑ Both	PSNR ↑ InVis	PSNR ↑ Vis	SSIM ↑ Both	SSIM ↑ InVis	SSIM ↑ Vis	Perc Sim ↓ Both	Perc Sim ↓ InVis	Perc Sim ↓ Vis	Replica [46] PSNR ↑	SSIM ↑	Perc Sim ↓
1	Im2Im [60]	15.87	16.20	15.97	0.53	0.60	0.48	2.99	0.58	2.05	17.42	0.66	2.29
2	Tatarchenko *et al*. [47]	14.79	14.83	15.05	0.57	0.62	0.53	3.73	0.74	2.50	14.36	0.68	3.36
3	Vox [42] w/ UNet	18.52	17.85	19.05	0.57	0.57	0.57	2.98	0.77	1.96	18.69	0.71	2.68
4	Vox [42] w/ ResNet	20.62	19.64	21.22	0.70	0.69	0.68	1.97	0.47	1.19	19.77	0.75	2.24
5	SynSin [53]	20.91	19.80	21.62	0.71	0.71	0.70	1.68	0.43	0.99	21.94	0.81	1.55
6	ours w/o inpainting	–	–	25.42	–	–	0.80	–	–	0.68	–	–	–
7	ours	**24.67**	**22.90**	**26.00**	**0.82**	**0.77**	**0.82**	**1.05**	**0.35**	**0.54**	**23.51**	**0.85**	**1.32**

Table 1: Novel View Synthesis: Performance of our and previous approaches on the Matterport dataset and the Replica dataset. All models are trained on Matterport and evaluated on both datasets. See Sec. 4.1 for details.

We build our inpainting network based on the generator in pix2pixHD [52], which translates a 4-channel input (the rendered image I_{out} and its foreground mask F_{out}) into a 3-channel output image I_{paint}. Our inpainting network outputs an entire image – it not only fills the invisible regions but also refines the image details in the visible regions. We apply the same RGB pixel L1 loss L^{rgb}_{paint} and perceptual loss L^{pc}_{paint} as in Sec. 3.3 on the inpainting output I_{paint}.

Training. We train our model using the Adam optimizer [17] with a weighted combination of losses as $L = \lambda_1 L^{rgb}_{out} + \lambda_2 L^{pc}_{out} + \lambda_3 L^{rgb}_{paint} + \lambda_4 L^{pc}_{paint} + \lambda_5 L_g + \lambda_6 L_m$ with $\lambda_1 = \lambda_3 = 8$, $\lambda_2 = \lambda_4 = 2$, $\lambda_5 = 0.2$, and $\lambda_6 = 10^{-4}$. Our model is trained for a total of 50000 iterations with batch size 64 and 10^{-4} learning rate.

We use a grid mesh with size $W_m \times H_m = 33 \times 33$ (and also 65×65 in Sec. 4.2). Following SynSin [53], we use $W_{im} \times H_{im} = 256 \times 256$ as the input and output image size. Our mesh implementation is based on PyTorch3D [33].

3.5. Extension: multi-layered Worldsheets

Although shrink-wrapping a single mesh sheet onto images works well on a wide range of scenes, one limitation is that it assumes that the foreground objects are connected to the background by mesh faces, which sometimes causes artifacts near object boundaries or depth discontinuities.

We propose an extension to address this limitation: predicting and warping multiple layers of Worldsheet onto the scene, where each sheet has a transparency channel in its texture map, loosely inspired by layered-depth images [39]. This allows some layers to fit the foreground object and others to capture the background. Specifically, we predict grid offset and depth for each mesh sheet from the feature map $\{q_{w,h}\}$ following Eqn. 1 to 3 with separate parameters. We also predict an $H_{im} \times W_{im}$ alpha map for each sheet using a deconvolution layer on $\{q_{w,h}\}$, which is then projected to the transparency channel in the UV texture map of the associated sheet. Finally, the multiple mesh sheets are rendered in the novel view using alpha compositing [44]. The whole model can be trained end-to-end under the same supervision. In Sec. 4.4, we find that, qualitatively, this extension leads to better handling of occlusions and parallax effect than a single mesh sheet.

4. Experiments

We evaluate our model on three datasets: Matterport [3], Replica [46], and RealEstate10K [59], following the experimental setup and details from [53]. We then provide analysis on in-the-wild images and multi-layered sheets.

4.1. Evaluation on Matterport and Replica

We first train and evaluate our approach on the Matterport dataset [3], which contains 3D scans of homes. We load the Matterport dataset in the Habitat simulator [36], following the same training, validation, and test splits as in SynSin [53]. During training, we supervise our model with paired 2D images of the input and the target views. We empirically find that it works slightly better to first train the scene mesh predictor (Sec. 3.1) and then freeze the scene mesh to further train the inpainting network (Sec. 3.4), rather than training both components jointly from scratch.

Metrics. Following SynSin [53], we evaluate the predicted novel view images I_{paint} using three metrics: Peak Signal-to-Noise Ratio (**PSNR**; higher is better), Structural Similarity (**SSIM**; higher is better), and Perceptual Similarity distance (**Perc Sim**; lower is better). The Perc Sim metric is based on the convolutional feature distance between the prediction and the ground-truth, which is shown to be highly correlated with human judgement [58, 53]. Since only a part of the target view image can be seen from the input image as illustrated in Figure 3, we separately evaluate these metrics on visible regions (**Vis**, which can be seen from the input view), invisible regions (**InVis**, which cannot be seen and must be imagined), and the entire image (**Both**). Note that the visible region masks are obtained from the ground-truth scene geometry and camera frustum (available from the Habitat simulator) instead of predicted by our mesh, and are the same as in SynSin's evaluation.

Baselines. We compare our method to several previous approaches: **Im2Im** [60] is an image-to-image translation method which predicts an appearance flow to warp an input view to the target view based on an input camera trans-

Figure 4: Novel views from our and previous methods on the Matterport dataset (scene mesh shown in the last column). The first row have same viewpoint change as in [53] while the second row has 2× larger camera angle change.

		Matterport [3] (2× cam. change)									Replica [46] (2× cam. change)		
		PSNR ↑			SSIM ↑			Perc Sim ↓			PSNR ↑	SSIM ↑	Perc Sim ↓
#	Method	Both	InVis	Vis	Both	InVis	Vis	Both	InVis	Vis			
1	Im2Im [60]	14.93	15.16	15.28	0.51	0.56	0.46	3.26	0.93	1.91	15.91	0.63	2.63
2	Tatarchenko *et al.* [47]	14.71	14.77	15.08	0.56	0.61	0.52	3.74	1.04	2.14	14.19	0.68	3.37
3	SynSin [53]	19.15	17.76	20.69	0.67	0.66	0.66	2.06	0.78	0.96	19.63	0.77	1.94
4	ours w/o inpainting	–	–	24.20	–	–	0.76	–	–	0.69	–	–	–
5	ours	**22.62**	**20.89**	**24.76**	**0.77**	**0.72**	**0.77**	**1.41**	**0.63**	**0.56**	**21.12**	**0.81**	**1.70**

Table 2: Novel View Synthesis: Generalization performance to larger viewpoint changes of our model vs. previous approaches on the Matterport dataset and the Replica dataset. All models are trained on the Matterport dataset and evaluated on both datasets with 2× larger camera angle changes than in the training data. See Sec. 4.1 for details.

formation. **Tatarchenko *et al.*** [47] is similar to Im2Im, but directly predicts the target view image instead of an appearance flow. **Vox w/ UNet** and **Vox w/ ResNet** are two variants of the deep voxel representation [42] with different encoder-decoder architectures based on UNet, or ResNet as implemented in [53]. **SynSin** [53] projects a dense feature point cloud (extracted from every image pixel) to the target camera pose and applies a refinement network on the point cloud projection to output the target view image.

We also evaluate the prediction of our model before inpainting (*i.e.* directly using the mesh rendering output I_{out} as the target view) to analyze how well our method performs with texture sampling and mesh rendering alone.

Results. The results are shown in Table 1. Even without inpainting, the mesh rendering output I_{out} from our method already outperforms previous approaches by a large margin under all the three metrics on the visible regions. With the help of an inpainting network, our final output I_{paint} has significantly higher performance than previous work on both invisible and visible regions, achieving a new state-of-the-art performance on this dataset. Figure 4 shows view synthesis examples from our method and previous work on the Matterport dataset, where our method can paint things such as doorframe or sofa at more precise locations.

Figure 5: Generalization of our model (trained on Matterport) to the Replica dataset without retraining (Sec. 4.1).

Generalization to the Replica dataset. Following [53], we also evaluate how well our model generalizes to another scene dataset, Replica [46], which contains high-quality laser scans of both homes and offices. We take our model trained on the Matterport dataset and directly evaluate on the Replica dataset without re-training. The results are shown in Table 1, where all methods are trained and evaluated under the same setting. It can be seen that our method achieves noticeably better generalization to this dataset and outperforms previous approaches by a large margin. Figure 5 shows view synthesis examples on the Replica dataset.

Generalization to larger viewpoint changes. We further analyze how well our and previous approaches generalize to larger camera pose changes beyond their training data. In this analysis, we sample new input-target view pairs on the test scenes with 2× larger camera angle changes than

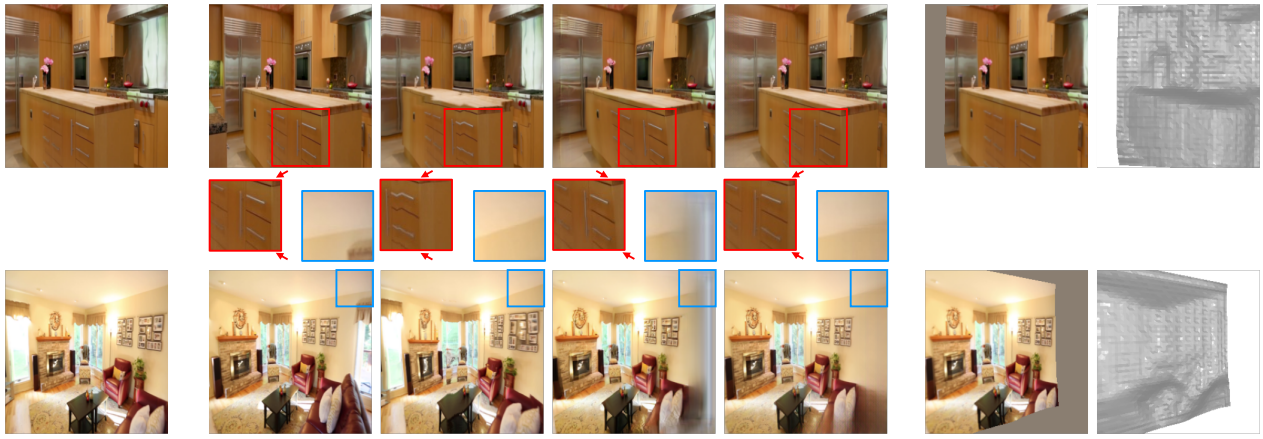


Figure 6: Novel views from our and previous methods on the RealEstate10K dataset (scene mesh shown in the last column).

#	Method	PSNR ↑	SSIM ↑	Perc Sim ↓
1	Im2Im [60]	17.05	0.56	2.19
2	Tatarchenko *et al.* [47]	11.35	0.33	3.95
3	Vox [42] w/ UNet	17.31	0.53	2.30
4	Vox [42] w/ ResNet	21.88	0.71	1.30
5	3DView (similar to [29])	21.88	0.66	1.52
6	SynSin [53]	22.83	0.75	1.13
7	Single-View MPI [48]	24.03	0.78	1.18
8	StereoMag [59]	25.34	**0.82**	1.19
9	ours (33×33 mesh)	26.24	**0.82**	0.83
10	ours (65×65 mesh)	**26.74**	**0.82**	**0.80**

Table 3: Comparison of our model with previous work on the RealEstate10K dataset [59]. See Sec. 4.2 for details.

#	Ablation setting	PSNR ↑	SSIM ↑	Perc Sim ↓
1	default (33×33 mesh reg. weight: 10^{-4})	26.24	**0.82**	0.83
2	reg. weight: 0	25.78	0.81	0.86
3	reg. weight: 10^{-5}	26.18	**0.82**	0.83
4	reg. weight: 10^{-3}	24.83	0.78	0.96
5	5×5 mesh	24.39	0.79	0.99
6	9×9 mesh	25.10	0.80	0.92
7	17×17 mesh	25.91	0.81	0.84
8	65×65 mesh†	**26.74**	**0.82**	**0.80**

Table 4: Ablations on the RealEstate10K dataset [59]. See Sec. 4.2 for details. (†: initialized from the default model.)

in the training data, and directly evaluate all approaches on these new viewpoints without retraining. The results are shown in Table 2, where our method largely outperforms other approaches under all metrics. Figure 4 (second row) shows an example under $2\times$ larger camera angle change.

4.2. Evaluation on RealEstate10K

The RealEstate10K dataset [59] consists of both indoor and outdoor scenes extracted from YouTube videos of houses. The input view and the target view are different video frames within a time range, with camera poses estimated using structure-from-motion.

On this dataset, we follow the experimental setup in SynSin [53] and use the same training, validation, and test data. In addition to using a 33×33 mesh, we also train our model with a higher resolution $W_m \times H_m = 65 \times 65$ mesh, which is initialized from a trained 33×33 mesh model with a new transposed convolution layer to upsample the feature map $\{q_{w,h}\}$ in Sec. 3.1 to 65×65 spatial dimensions.

We compare our method to several previous approaches. In addition to the baselines in Sec. 4.1, we also compared to three additional approaches. **3DView** is a system similar to the Facebook 3D Photo [29] based on layered depth images and is also a baseline in [53]. **Single-View MPI** [48] and **StereoMag** [59] both use multiplane images (MPIs), where Single-View MPI builds MPIs from a single input image while StereoMag relies on a stereo pair using images from two different views as input at test time. Except for StereoMag, all other methods use a single view at test time.

Results. We follow the evaluation protocol of [53] on RealEstate10K, with results[1] shown in Table 3. It can be seen that our method achieves the highest performance, outperforming previous approaches by a noticeable margin. Besides, a higher resolution 65×65 mesh gives a further performance boost. Figure 6 shows predicted novel views on this dataset. In addition, we visualize the pixel-wise squared error map on the prediction from our method and SynSin [53] in Figure 7, where our method paints objects at more precise locations compared to SynSin, resulting in higher PSNR and better quality.

Ablations. Since our model relies on deforming a mesh sheet, we first analyze the impact of geometric regularization on the mesh deformation. In Table 4, line 2 to 4 vary the weight of the mesh Laplacian regularization term L_m from its default value 10^{-4}. Comparing these variants to line 1,

[1]To compare with StereoMag [59] that uses two input views, in the evaluation protocol of SynSin [53] on RealEstate10K, the best metrics of two separate predictions based on each view were reported for single-view methods. We follow this evaluation protocol for consistency with [53] on RealEstate10K in Table 3 and 4. We also report averaged metrics over all predictions in supplemental, where the trends are consistent.

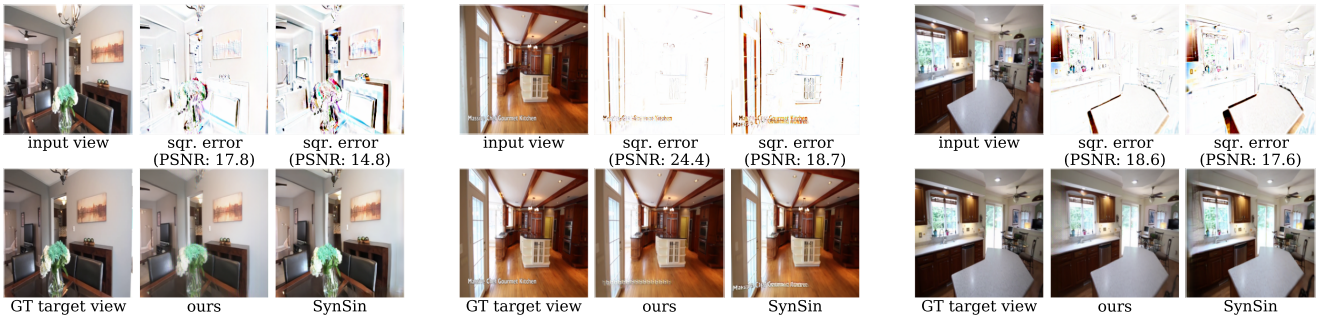


Figure 7: Squared error maps in target view from our method and SynSin [53] on the RealEstate10K dataset (darker is higher error). Our method paints things in the novel view at more precise locations, resulting in lower error and higher PSNR.

a higher regularization (10^{-3}, line 4) restricts the model's capacity to precisely fit the scene geometry and hence hurts the performance. Meanwhile, there is only a smaller drop when decreasing this regularization weight to 10^{-5} or even zero, suggesting that our differentiable rendering pipeline provides robust wrapping of the mesh onto the scene.

We further study the impact of mesh resolution $W_m \times H_m$ in line 5 to 8. As expected, higher mesh resolution allows fitting more fine-grained scene details and gives higher view synthesis performance, with the final 65×65 mesh giving the best performance. In addition, we find that with enough mesh resolution (such as 65×65), one can restrict the grid offset in Eqn. 1 and 2 to zero and only use the predicted depth in Eqn. 3 to deform the mesh, which gives only -0.13 PSNR drop (however, the grid offset makes a larger difference in lower-resolution meshes such as Figure 2).

4.3. Analysis: testing the limits of wrapping sheets

So far, we have shown that the idea of wrapping a mesh sheet onto an image achieves strong performance across all benchmarks. But one might wonder, how good is a planar sheet prior for novel view synthesis in any arbitrary images? To test the limits of wrapping a mesh *sheet*, we test it over a large variety of images including outdoor scenes, outdoor objects, indoor scenes, indoor objects, and even artistic paintings. We analyze our underlying mesh data structure by pretraining depth [21] to fill in the z values, and examine how well it generates novel views in-the-wild. As shown in Figure 1 (top row), although missing a few details (such as tree branches), a scene mesh sheet captures the geometric structures sufficient enough to render high-resolution (960×960) photorealistic novel views even from very large viewpoint changes. Please see videos at `worldsheet.github.io` for animation of continuously generated views. This result confirms that our mesh sheet data structure and warping procedure, despite being simple, are flexible enough to handle the variety of the visual world.

4.4. Analysis: multi-layered Worldsheets

As described in Sec. 3.5, to better handle sharp depth discontinuities, we explore the extension to stack multiple mesh sheet layers, so that foreground objects and the background can be placed on different layers. Interestingly, we observe that this extension does not make a noticeable improvement in RealEstate10K evaluation metrics compared to a single sheet (26.62 vs. 26.74 in PSNR) suggesting that single Worldsheet is sufficient enough for view synthesis application. However, qualitatively speaking, we notice that it allows better handling of occlusions and parallax effect in our model under large viewpoint changes, as shown in Figure 8. Please see supplemental for more details.

Figure 8: Using multiple mesh sheets (Sec. 4.4), our model (rightmost) better handles occlusions and generates parallax effect (*e.g.* sofa occluding the handrail when moving camera to the right) compared to SynSin [53] or a single sheet.

5. Conclusion

In this work, we propose Worldsheet, which synthesizes novel views from a single image by shrink-wrapping the scene with a grid mesh. Our approach jointly learns the scene geometry and generates novel views through differentiable texture sampling and mesh rendering, supervised with only 2D images of the input and the target views. The approach is category-agnostic and end-to-end trainable, resulting in state-of-the-art performance on single-image view synthesis across three datasets by a large margin.

Acknowledgments. We are grateful to Alyosha Efros, Angjoo Kanazawa, Shubham Goel, Devi Parikh, Ross Girshick, Georgia Gkioxari, Justin Johnson, Brian Okorn and other colleagues at FAIR and CMU for fruitful discussions. This work was supported in part by DARPA Machine Common Sense grant (associated with D. Pathak and not associated with Facebook Inc).

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. *arXiv preprint arXiv:1906.08240*, 2019. 2

[2] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999. 2

[3] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*. IEEE, 2017, License for Matterport dataset available at http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf. 2, 5, 6, 12

[4] Shenchang Eric Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995. 2

[5] Xu Chen, Jie Song, and Otmar Hilliges. Monocular neural image based rendering with continuous view control. In *ICCV*, 2019. 2

[6] Antonio Criminisi, Ian Reid, and Andrew Zisserman. Single view metrology. *International Journal of Computer Vision*, 2000. 2

[7] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2

[8] Frank Dellaert. Nerf explosion 2020. https://dellaert.github.io/NeRF. 2

[9] Ruofei Du, David Li, and Amitabh Varshney. Project geollery. com: Reconstructing a live mirrored world with geotagged social media. In *The 24th International Conference on 3D Web Technology*, pages 1–9, 2019. 2

[10] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoint without keypoints. In *ECCV*, 2020. 3, 4

[11] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 3

[13] Derek Hoiem, Alexei A Efros, and Martial Hebert. Automatic photo pop-up. In *ACM SIGGRAPH 2005 Papers*. 2005. 1, 2

[14] Youichi Horry, Ken-Ichi Anjyo, and Kiyoshi Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997. 2

[15] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, pages 371–386, 2018. 2, 3, 4

[16] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018. 2, 3, 4

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[18] Johannes Kopf, Kevin Matzen, Suhib Alsisan, Ocean Quigley, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, Matthew Yu, et al. One shot 3d photography. *ACM Transactions on Graphics (TOG)*, 2020. 2

[19] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. In *ICCV*, 2019. 2, 3

[20] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015. 2

[21] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv preprint arXiv:1907.01341*, 2019. 8

[22] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2

[23] Yikai Li, Jiayuan Mao, Xiuming Zhang, Bill Freeman, Josh Tenenbaum, Noah Snavely, and Jiajun Wu. Multi-plane program induction with 3d box priors. In *Advances in Neural Information Processing Systems*, 2020. 2

[24] David Liebowitz, Antonio Criminisi, and Andrew Zisserman. Creating architectural models from images. In *Computer Graphics Forum*. Wiley Online Library, 1999. 2

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1

[26] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 2

[27] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *ICCV*, 2019. 2, 3, 4

[28] Madara Lukjanovica. Italy alleyway sketch. https://www.pinterest.com/pin/546342998522093232. 1

[29] 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. 7

[30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020. 2

[31] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Transactions on Graphics (TOG)*, 2019. 2

[32] Georgios Pavlakos, Nikos Kolotouros, and Kostas Daniilidis. Texturepose: Supervising human mesh estimation with texture consistency. In *ICCV*, 2019. 3

[33] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 2, 3, 4, 5, 14

[34] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *ECCV*, 2020. 2

[35] Gernot Riegler and Vladlen Koltun. Stable view synthesis. *arXiv preprint arXiv:2011.07233*, 2020. 2

[36] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *ICCV*, 2019. 5

[37] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 2008. 2

[38] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006. 2

[39] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, 1998. 5

[40] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *CVPR*, 2020. 2

[41] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009. 3, 4

[42] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019. 5, 6, 7, 12

[43] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019. 2

[44] Alvy Ray Smith. Image compositing fundamentals. *Microsoft Corporation*, 1995. 5

[45] 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 *CVPR*, 2019. 2

[46] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 2, 5, 6, 12

[47] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *ECCV*. Springer, 2016. 2, 5, 6, 7, 12

[48] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *CVPR*, 2020. 2, 7, 12

[49] University of Oxford. Oxford New College. https://www.biology.ox.ac.uk/applications. 1

[50] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *ECCV*. Springer, 2014. 2

[51] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. *arXiv preprint arXiv:2102.13090*, 2021. 2

[52] 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 *CVPR*, 2018. 4, 5

[53] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *CVPR*, pages 7467–7477, 2020. 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 17

[54] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[55] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Interpretable transformations with encoder-decoder networks. In *ICCV*, 2017. 2

[56] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *CVPR*, 2020. 2

[57] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *ICLR*, 2016. 3

[58] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 5

[59] 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. 2, 5, 7, 11, 12

[60] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *ECCV*. Springer, 2016. 5, 6, 7, 12

[61] 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. 2

Worldsheet: Wrapping the World in a 3D Sheet for View Synthesis from a Single Image
(Supplementary Material)

A. Continuous and large viewpoint changes

Our approach allows synthesizing continuous novel views by smoothly moving to a new camera pose that is largely different from the input. We kindly request the readers to view the videos at `worldsheet.github.io` to better understand the performance of our method. In these videos, we compare our synthesized novel views (from a single image) to SynSin [53] on the RealEstate10K dataset with simulated large view-point changes (the first frame contains the input view and the rest of the frames are synthesized). From the videos, it can be seen that our model can generate novel views with much larger camera translation and rotation than in the training data, while SynSin often suffers from severe artifacts in these cases, likely because its refinement network does not generalize well to a sparser point cloud (resulting from large camera zoom-in or rotation).

We also show in these videos continuously synthesized novel views on high resolution (960×960) images over a wide range of scenes (the first frame is the input view), as described in our analysis in Sec. 4.3 in the main paper.

B. Ablation study: using depth supervision

In our experiments in the paper, we show that our model can be trained using only two views of a scene *without* 3D or depth supervision. In this section, we further analyze our approach by training it *with* depth supervision on the Matterport dataset, where the ground-truth depth can be obtained from the Habitat simulator.

In this analysis, we modify the differentiable mesh renderer to render RGB-D images from our mesh, and apply an L1 loss between the ground-truth and the rendered depth as additional supervision. We also compare with the performance of SynSin with depth supervision (reported in [53]). The results are shown in Table B.1. It can be seen that our model without depth supervision (the default setting; line 7) works almost equally as well as its counterpart using depth supervision (line 8) on the Matterport dataset and generalizes better to the Replica dataset. In addition, it outperforms SynSin under both supervision settings (lines 5-6).

C. Additional analyses on RealEstate10K

As described in Sec. 4.2 in the main paper, we follow the evaluation protocol of SynSin [53] on the RealEstate10K dataset. To enable comparison with StereoMag [59] that uses two input views on this dataset, in [53], the best metrics of two views were reported for single-view methods. At test time, for each target view, this involves making two separate predictions based on two different input views respectively, and then selecting the best metrics between the two predictions as the score for this target view. Note that this evaluation protocol is only applied to the RealEstate10K dataset (in Table 3 and 4 in the main paper) and is not applied to Matterport or Replica.

In this section, we further evaluate by taking the average metrics over all predictions to measure how well the model does on average from a single input view and to be consistent with our evaluation on Matterport and Replica in Table 1 and 2 in the main paper. Apart from metrics over the entire image, we would also like to analyze how well each model does on rendering regions seen in the input view vs. invisible regions (where things must be imagined). However, we cannot compute the exact visibility map as there are no ground-truth geometry annotations in the RealEstate10K dataset. To get an approximation, we evaluate on the central $\frac{W_{im}}{2} \times \frac{H_{im}}{2} = 128 \times 128$ crop of the target image (**Center**, which is nearly always visible) and the rest of the image (**Peripheral**, containing most of the invisible regions).

The results of these analyses are shown in Table C.1. It can be seen that our method achieves the highest performance on both the center regions (which are mostly visible) and the peripheral regions, outperforms the previous single-view based approaches by a large margin.

Our model uses a simple ResNet-50 backbone with an output stride of 8 pixels to extract image features (see Sec. 3.1 in the main paper), while SynSin [53] adopts a U-Net backbone that has a higher output feature resolution same as the input image (*i.e.* output feature stride is 1 pixel). To further study the impact of different backbone architectures, we train a variant of SynSin by replacing its U-Net backbone with the same ResNet-50 backbone pretrained on ImageNet (and upsampling its output feature map to stride 1 with a deconvolution layer) to be consistent with our model, shown in line 4 in Table C.1. Comparing it with line 3 or 6, it can be seen that this variant of SynSin with ResNet-50 backbone performs worse than the default SynSin architecture, or our model. This suggests that SynSin requires a high-resolution feature output to build a per-pixel point-cloud for view synthesis, while our model is able to work with a lower resolution (a larger stride) in the backbone.

#	Method	Matterport [3] PSNR ↑ Both	PSNR ↑ InVis	PSNR ↑ Vis	SSIM ↑ Both	SSIM ↑ InVis	SSIM ↑ Vis	Perc Sim ↓ Both	Perc Sim ↓ InVis	Perc Sim ↓ Vis	Replica [46] PSNR ↑	SSIM ↑	Perc Sim ↓
1	Im2Im [60]	15.87	16.20	15.97	0.53	0.60	0.48	2.99	0.58	2.05	17.42	0.66	2.29
2	Tatarchenko *et al*. [47]	14.79	14.83	15.05	0.57	0.62	0.53	3.73	0.74	2.50	14.36	0.68	3.36
3	Vox [42] w/ UNet	18.52	17.85	19.05	0.57	0.57	0.57	2.98	0.77	1.96	18.69	0.71	2.68
4	Vox [42] w/ ResNet	20.62	19.64	21.22	0.70	0.69	0.68	1.97	0.47	1.19	19.77	0.75	2.24
5	SynSin [53]	20.91	19.80	21.62	0.71	0.71	0.70	1.68	0.43	0.99	21.94	0.81	1.55
6	SynSin (w/ depth sup.) [53]	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	ours	**24.67**	**22.90**	**26.00**	**0.82**	**0.77**	**0.82**	**1.05**	**0.35**	**0.54**	**23.51**	**0.85**	**1.32**
8	ours (w/ depth sup.)	24.75	22.85	26.18	0.82	0.77	0.82	1.06	0.36	0.54	22.78	0.84	1.51

Table B.1: Analyses of our model and SynSin using explicit depth supervision during training (line 8 and 6) on the Matterport dataset. Our model without depth (the default setting; line 7) performs almost equally as well as its counter part with depth supervision (line 8) on Matterport and generalizes better to Replica, outperforming both variants of SynSin (line 5 and 6). See Sec. B for details.

#	Method	RealEstate10K [59] (averaged metrics over all predictions) PSNR ↑ Both	PSNR ↑ Peripheral	PSNR ↑ Center	SSIM ↑ Both	SSIM ↑ Peripheral	SSIM ↑ Center	Perc Sim ↓ Both	Perc Sim ↓ Peripheral	Perc Sim ↓ Center
1	Im2Im [60]	15.56	15.65	15.80	0.51	0.53	0.44	2.59	1.93	0.65
2	Tatarchenko *et al*. [47]	11.11	11.32	10.68	0.32	0.35	0.24	3.96	2.96	1.13
3	SynSin [53]	20.47	20.35	21.65	0.68	0.68	0.68	1.49	1.16	0.29
4	SynSin w/ R-50 backbone†	19.37	19.33	20.18	0.65	0.65	0.64	1.64	1.26	0.34
5	Single-View MPI [48]	21.17	20.90	23.03	0.70	0.70	0.71	1.59	1.27	0.31
6	ours (33×33 mesh)	23.11	22.90	24.83	**0.75**	0.74	**0.76**	1.17	0.94	**0.21**
7	ours (65×65 mesh)	**23.41**	**23.20**	**25.17**	**0.75**	**0.75**	**0.76**	**1.14**	**0.92**	**0.21**

Table C.1: Performance of our and previous approaches on the RealEstate10K dataset with metrics averaged over all predictions, which is consistent with our evaluation on Matterport and Replica in Sec. 4.1 in the main paper. We evaluate on the entire 256×256 image (**both**), the **center** 128×128 crop (nearly always visible), and the remaining **peripheral** regions (containing most of the invisible regions). See Sec. C for details. (†: We also evaluate a variant of SynSin by replacing its U-Net backbone with the same ResNet-50 backbone pretrained on ImageNet as used in our model.)

D. Details on differentiable texture sampler

Our differentiable texture sampler (Sec. 3.2 in the main paper) splats image pixels onto the texture map through each face. This splatting procedure involves three main steps: forward-mapping, normalization, and hole filling.

Suppose the flow $(u, v) = f(i, j)$ maps image coordinates (i, j) to UV coordinates (u, v) on the texture map, which can be obtained through mesh rasterization (here we drop the face index k for simplicity). To implement $\hat{T} = \text{splat}(I_{in}, f)$ (Eqn. 5 in the main paper), we first forward-map the image pixels to the texture map, using bilinear assignment when the mapped texture coordinates (u, v) do not fall on integers, as forward_map below.

```
function T = forward_map(I, f)
  T = all_zeros(W_uv, H_uv)
  for i in 1:W_im
    for j in 1:H_im
      u, v = f(i, j)
      uf, uc = floor(u), ceil(u)
      vf, vc = floor(v), ceil(v)
      T[uf,vf] += I[i,j] * (uc - u) * (vc - v)
      T[uc,vf] += I[i,j] * (u - uf) * (vc - v)
      T[uf,vc] += I[i,j] * (uc - u) * (v - vf)
      T[uc,vc] += I[i,j] * (u - uf) * (v - vf)
    end
  end
  return T
```

In the implementation above, gradients can be taken over f through the bilinear weights.

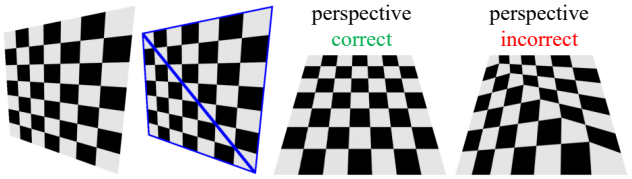
However, forward mapping alone will lead to incorrect pixel intensity (*e.g.* imagine down-scaling an image to half its width and height by forward-mapping – each pixel in the low-resolution image will receive assignment from 4 pixels and become $4\times$ brighter). Hence, a second normalization step is applied:

$$\hat{T}_{sum} = \text{forward\_map}(I_{in}, f) \quad \text{(D.1)}$$

$$\hat{W}_{sum} = \text{forward\_map}(I_{one}, f) \quad \text{(D.2)}$$

$$\hat{T}_{norm} = \hat{T}_{sum} / \max\left(\hat{W}_{sum}, 10^{-4}\right) \quad \text{(D.3)}$$

where I_{one} is an $W_{im} \times H_{im}$ image with all ones as its pixel



(a) input view (b) 2×2 mesh (c) novel view via our texture sampler (d) novel view via vertex *uv* on screen

Figure D.1: Our texture sampler is perspective-correct.

intensity. $\hat{T}_{norm}$ contains the normalized splatting result. A threshold 10^{-4} is applied to avoid division by zero (which could happen due to holes described below).

Filling holes with Gaussian filtering. It is well known that the bilinear forward mapping above often leads to holes in the output (*e.g.* imagine up-scaling an image to a much larger size – there will be gaps in the output image as some pixels will not receive assignments). To minimize hole occurrence, in our UV texture map we assign the UV coordinates of each mesh vertex with an equally-spaced $W_m \times H_m$ lattice grid, and use the same image size as the texture map size ($W_{uv} \times H_{uv} = W_{im} \times H_{im}$). This ensures that most texels on the texture map receive assignments in forward mapping, so that holes rarely occur in $\hat{T}_{norm}$. However, to address corner cases, we further apply a Gaussian filter to fill the holes in $\hat{T}_{norm}$ (where $\hat{W}_{sum}$ is zero as no assignment is received from forward-mapping):

$$\hat{M} = \mathrm{I}\left[\hat{W}_{sum} > 0\right] \tag{D.4}$$

$$\hat{T}_g = \left(F_g * \hat{T}_{norm}\right) / \left(F_g * \hat{M}\right) \tag{D.5}$$

$$\hat{T} = \hat{M} \cdot \hat{T}_{norm} + \left(\left(1 - \hat{M}\right) \cdot \hat{T}_g\right) \tag{D.6}$$

where F_g is a discrete 2D Gaussian kernel for image filtering (we use kernel size 7 and standard deviation 2 for F_g in our implementation). Here $\hat{M}$ is a binary mask indicating which pixels have received assignments in forward mapping (*i.e.* 1 means valid and 0 means holes), $\hat{T}_g$ is the Gaussian-blurred version of $\hat{T}_{norm}$ (where the division ensures the correct pixel intensity; otherwise it will be darker due to holes in $\hat{T}_{norm}$) and is used to fill only the holes in $\hat{T}_{norm}$. We use $\hat{T}$ in Eqn. D.6 as the final splatting output.

Perspective correctness. A main purpose of our differentiable texture sampler is to build perspective-correct novel views during texture reconstruction. We note that the alternative solution of directly using the input image as a texture map by putting vertex uv texture coordinates in the input screen space for mesh rendering breaks perspective correctness, as shown in Figure D.1 (d). For perspective-correct novel views, one needs to invert the texture-map-to-image perspective transform when building UV texture maps from the image, which we implement in our texture sampler shown in Figure D.1 (c).

E. Details on multi-layered Worldsheets

In our proposed Worldsheet model, we build a scene mesh by warping a planar sheet onto the scene. This model is capable of handling moderate occlusion and generating plausible novel views by deforming the mesh along object boundaries and refining the predicted novel view with an inpainting network. However, we also acknowledge that artifacts can sometimes occur in occluded regions or object boundaries when the disparity is very large between the input view and the novel view. This is partly because our current approach of deforming a mesh onto the scene does not capture all the fine-grained geometric details (such as the flower boundary as shown in the last two failure cases in the supplemental videos. We believe there is room for improvement in this direction (*e.g.* via adaptive resolution), which we are interested in exploring in future work.

In Sec. 3.5 in the main paper, we propose a simple extension with multi-layered Worldsheets. The main purpose of this extension is to separate objects or scene structures at different depth levels into different mesh layers, instead of placing them all on a single sheet. In this extension, the 3D mesh geometry of each layer provides the geometric support for the scene components, while the transparency channel in the RGBA texture maps allows segmentation between different components. For example, to represent a sofa object, a mesh layer can be wrapped onto a larger 3D surface region covering the sofa surface, with its texture map containing the sofa texture over the object region while being transparent on the surrounding regions.

Specifically, we predict and warp a total of L mesh sheets (*i.e.* L layers) onto the scene for view synthesis. For each layer $l = 1, \cdots, L$, we predict its grid offset $\left(\Delta\hat{x}^{(l)}_{w,h}, \Delta\hat{y}^{(l)}_{w,h}\right)$ and its depth $z^{(l)}_{w,h}$ from the convolutional feature map $\{q_{w,h}\}$ similar to Sec. 3.1, and also predict a pixel-wise transparency map $\alpha^{(l)}$ of size $H_{im} \times W_{im}$ in the screen space of the input view as follows.

$$\Delta\hat{x}^{(l)}_{w,h} = \frac{\tanh\left(W^{(l)}_1 q_{w,h} + b^{(l)}_1\right)}{W_m - 1} \tag{E.1}$$

$$\Delta\hat{y}^{(l)}_{w,h} = \frac{\tanh\left(W^{(l)}_2 q_{w,h} + b^{(l)}_2\right)}{H_m - 1} \tag{E.2}$$

$$z^{(l)}_{w,h} = g\left(W^{(l)}_3 q_{w,h} + b^{(l)}_3\right) \tag{E.3}$$

$$\left\{\alpha^{(l)}_{i,j}\right\} = \sigma\left(\mathrm{deconv}\left(\{q_{w,h}\}; W^{(l)}_4, b^{(l)}_4\right)\right) \tag{E.4}$$

Here $\alpha^{(l)}_{i,j}$ is the scalar alpha (*i.e.* transparency) value at image pixel (i, j), deconv is a deconvolution (*i.e.* transposed convolution) layer over the feature map with an output size equal to the image size, and $\sigma(\cdot)$ is the sigmoid function to transform the alpha values to the range between 0 and 1.

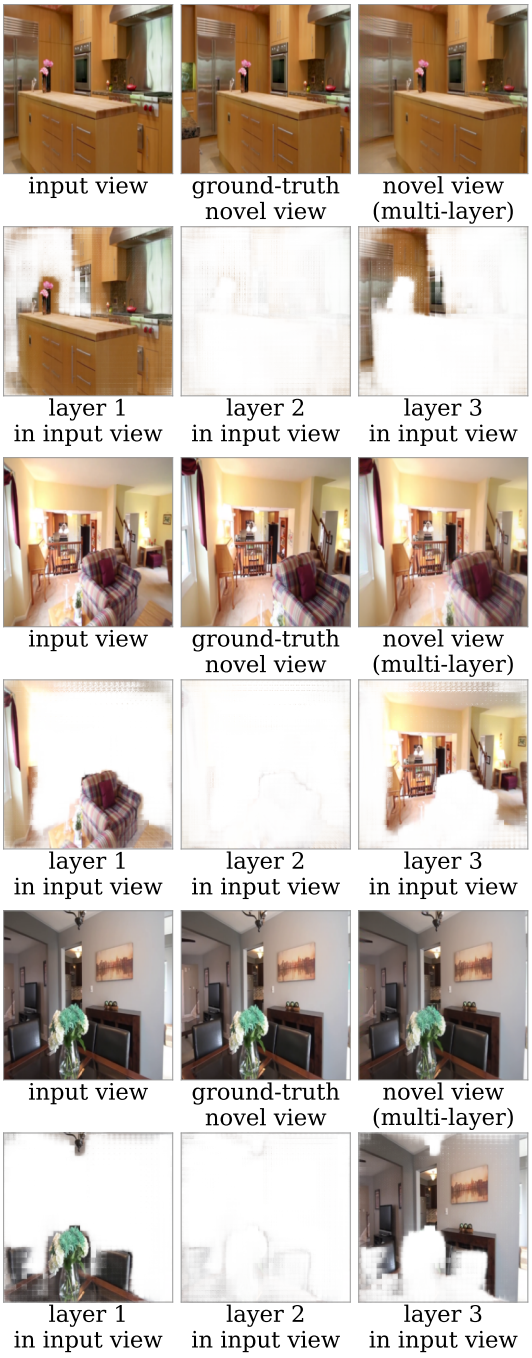


Figure E.1: View synthesis examples on RealEstate10K from our extension on multi-layered Worldsheets (see Sec. E for details). The top rows show the predicted novel views, while the bottom rows visualize each mesh layer $l = 1, ..., L$ (here $L = 3$) along with its RGBA texture map in the input view (white is transparent). Through end-to-end training with only 2D rendering losses, the model learns to place scene structures at different depth levels onto different mesh layers, and separate foreground objects (*e.g.* kitchen counter, sofa, or table) from their background.

For each layer $l = 1, \cdots, L$, we construct a corresponding 3D mesh sheet $M^{(l)}$ following the procedure in Sec. 3.1 and also build its UV texture map $\hat{T}^{(l)}$ consisting of RGBA channels by concatenating the predicted transparency values $\alpha_{i,j}^{(l)}$ with the input image and splatting them onto the mesh texture space using our differentiable texture sampler in Sec. 3.2. Finally, we render all the mesh faces from all L layers $M^{(1)}, \cdots, M^{(L)}$ in the novel view along with their RGBA UV texture maps $\hat{T}^{(1)}, \cdots, \hat{T}^{(L)}$ through alpha compositing. The whole model can be trained end-to-end under the same supervision using only 2D rendering losses.

We use a total of $L = 3$ layers in our analyses. Figure E.1 visualizes this extension on multi-layered Worldsheets. It can be seen that through end-to-end training, the model learns to place scene structures at different depth levels onto different mesh layers and separate foreground objects (*e.g.* kitchen counter, sofa, or table) from their background. We qualitatively find that it better handles occlusions and parallax effect under large viewpoint changes, as shown in Sec. 4.4 in the main paper.

F. Hyper-parameters in our model

In our nonlinear function $g(\cdot)$ to scale the network prediction into depth values (in Eqn. 3 in the main paper), we use different output scales based on the depth range in each dataset. On Matterport and Replica, we use

$$g(\psi) = 1/(0.75 \cdot \sigma(\psi) + 0.01) - 1. \tag{F.1}$$

On RealEstate10K (which has larger depth range), we double the output depth scale and use

$$g(\psi) = 2/(0.75 \cdot \sigma(\psi) + 0.01) - 2. \tag{F.2}$$

However, we find that the performance of our model is quite insensitive to the hyper-parameters in $g(\cdot)$.

In our differentiable texture sampler and the mesh renderer, we mostly follow the hyper-parameters in PyTorch3D [33]. We use $K = 10$ faces per pixel and 1e-8 blur radius in mesh rasterization, 1e-4 sigma and 1e-4 gamma in softmax RGB blending, and background color filled with the mean RGB intensity on each dataset. On Matterport and Replica, the input views have 90-degree field-of-view. On RealEstate10K, we multiply the actual camera intrinsic matrix of each frame into its camera extrinsic R and T matrices, so that we can still use the same intrinsics and 90-degree field-of-view in the renderer. On high resolution images in the wild (Sec. 4.3 in the main paper), we assume 45-degree field-of-view.

We choose our mesh size W_m and H_m based on the image size. In our experiments on Matterport and Replica (Sec. 4.1 in the main paper), we use 256×256 input image resolution following SynSin [53], and use pixel stride 8 on the lattice grid sheet (from which our mesh is built),

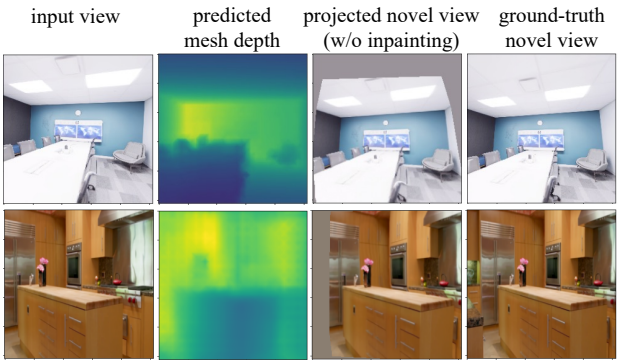


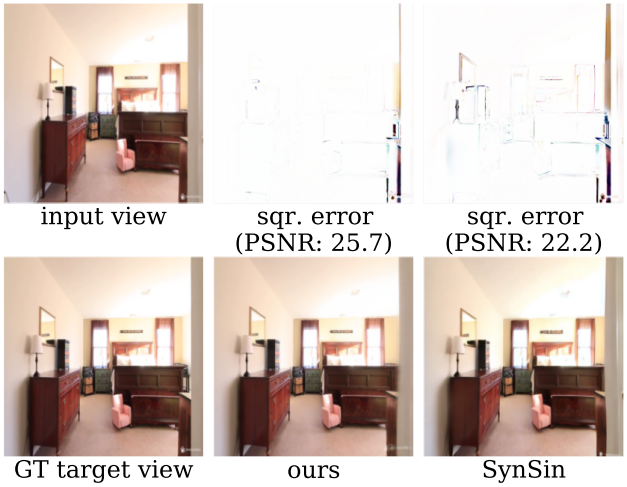
Figure G.1: Predicted depth maps from our scene mesh on Replica (upper) and RealEstate10K (lower).

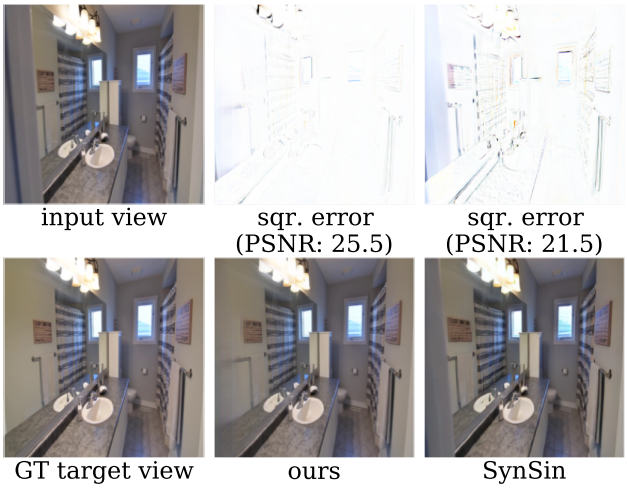
resulting in $W_m = H_m = 1 + 256/8 = 33$. On the RealEstate10K dataset (Sec. 4.2 in the main paper), we additionally experiment with pixel stride 4 on the grid sheet, giving $W_m = H_m = 1 + 256/4 = 65$. In our analysis on high resolution images in the wild (resized to have the image long side equal to 960 and padded to 960×960 square size for ease of rendering in PyTorch3D; Sec. 4.3 in the main paper), we use $W_m \times H_m = 129 \times 129$ mesh on the actual image regions (not including the padding regions).

G. More visualized examples

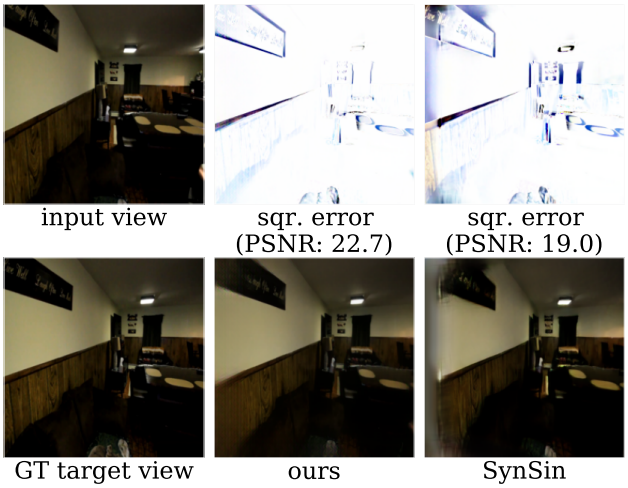
Figure G.1 shows the depth maps from our scene mesh, where most of the scene structure is captured, giving coherent novel view projections.

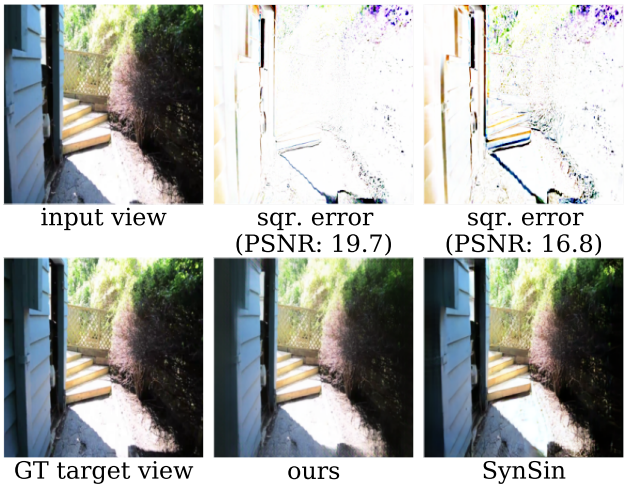
Figure G.2 shows additional visualization and error map comparisons between our approach and SynSin on the RealEstate10K dataset (similar to Figure 7 in the main paper), where our method paints things in the novel view at more precise locations with lower error and higher PSNR.


input view
sqr. error (PSNR: 25.7)
sqr. error (PSNR: 22.2)
GT target view
ours
SynSin


input view
sqr. error (PSNR: 25.5)
sqr. error (PSNR: 21.5)
GT target view
ours
SynSin


input view
sqr. error (PSNR: 24.5)
sqr. error (PSNR: 22.8)
GT target view
ours
SynSin


input view
sqr. error (PSNR: 22.7)
sqr. error (PSNR: 19.0)
GT target view
ours
SynSin


input view
sqr. error (PSNR: 19.7)
sqr. error (PSNR: 16.8)
GT target view
ours
SynSin

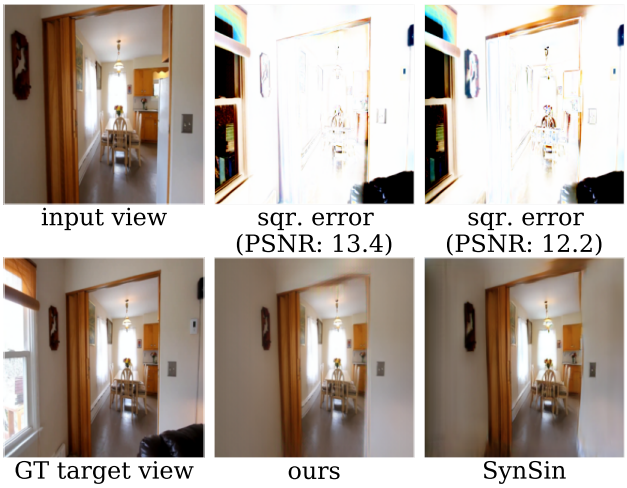

input view
sqr. error (PSNR: 13.4)
sqr. error (PSNR: 12.2)
GT target view
ours
SynSin

Figure G.2: Additional synthesized novel views (with squared error maps of the target view) from our method and SynSin [53] on the RealEstate10K dataset (darker is higher error). Our method paints things in the novel view at more precise locations, resulting in lower error and higher PSNR.