

3D Photography using Context-aware Layered Depth Inpainting

Meng-Li Shih¹²
shihsm1@gapp.nthu.edu.tw

Shih-Yang Su¹
shihyang@vt.edu

Johannes Kopf³
jkopf@fb.com

Jia-Bin Huang¹
jbhuang@vt.edu

¹Virginia Tech

²National Tsing Hua University

³Facebook

<https://shihmengli.github.io/3D-Photo-Inpainting>

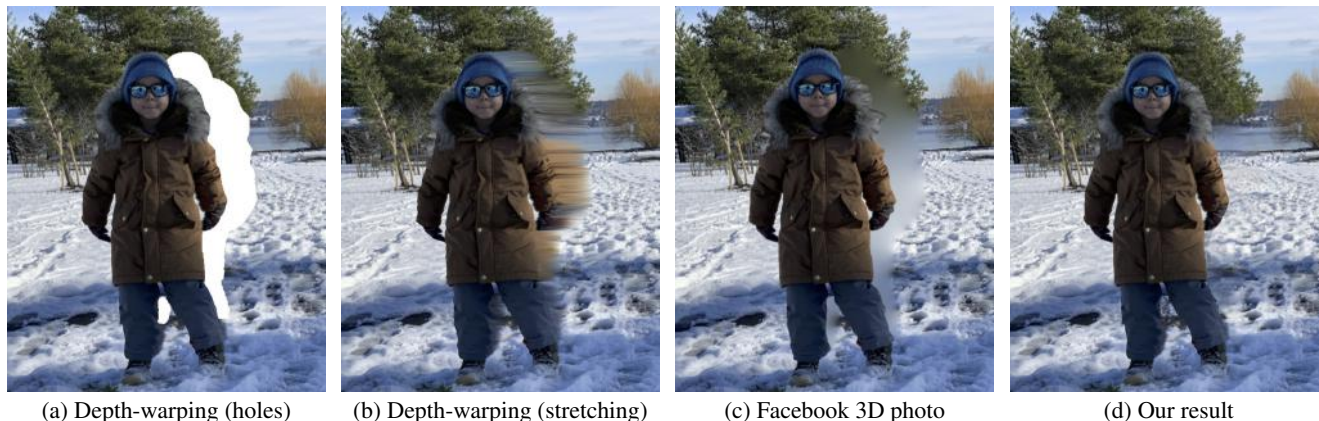


Figure 1. **3D photography from a single RGB-D image.** Naïve methods either produce holes (a) or stretch content (b) at disocclusions. Color and depth inpainting using diffusion is better, but provides a too smooth appearance (c). Our approach is capable of synthesizing new color/depth texture and structures, leading to more photorealistic novel views (d).

Abstract

We propose a method for converting a single RGB-D input image into a 3D photo — a multi-layer representation for novel view synthesis that contains hallucinated color and depth structures in regions occluded in the original view. We use a Layered Depth Image with explicit pixel connectivity as underlying representation, and present a learning-based inpainting model that synthesizes new local color-and-depth content into the occluded region in a spatial context-aware manner. The resulting 3D photos can be efficiently rendered with motion parallax using standard graphics engines. We validate the effectiveness of our method on a wide range of challenging everyday scenes and show less artifacts compared with the state of the arts.

1. Introduction

3D photography—capturing views of the world with a camera and using image-based rendering techniques for novel view synthesis—is a fascinating way to record and reproduce visual perception. It provides a dramatically more immersive experience than old 2D photography: almost lifelike in Virtual Reality, and even to some degree on normal flat displays when displayed with parallax.

Classic image-based reconstruction and rendering tech-

niques, however, require elaborate capture setups involving many images with large baselines [17, 64, 27, 47, 19, 12], and/or special hardware (e.g., Lytro Immerge, Facebook Manifold camera¹).

Recently, we have seen work to make capture for 3D photography more effortless by using cell phone cameras and lowering baseline requirements [17, 18]. In the most extreme cases, novel techniques such as Facebook 3D Photos² now just require capturing a single snapshot with a dual lens camera phone, which essentially provides an RGB-D (color and depth) input image.

In this work we are interested in rendering novel views from such an RGB-D input. The most salient features in rendered novel views are the disocclusions due to parallax: naïve depth-based warping techniques either produce gaps here (Figure 1a) or stretched content (1b). Recent methods try to provide better extrapolations.

Stereo magnification [77] and recent variants [56, 40] use a fronto-parallel multi-plane representation (MPI), which is synthesized from the small-baseline dual camera stereo input. However, MPI produces artifacts on sloped surfaces. Besides, the excessive redundancy in the multi-

¹<https://facebook360.fb.com/2018/05/01/red-facebook-6dof-camera/>

²<https://facebook360.fb.com/2018/10/11/3d-photos-now-rolling-out-on-facebook-and-in-vr/>

plane representation makes it memory and storage inefficient and costly to render.

Facebook 3D Photos use a layered depth image (LDI) representation [51], which is more compact due to its sparsity, and can be converted into a light-weight mesh representation for rendering. The color and depth in occluded regions are synthesized using heuristics that are optimized for fast runtime on mobile devices. In particular it uses an isotropic diffusion algorithm for inpainting colors, which produces overly smooth results and is unable to extrapolate texture and structures (Figure 1c).

Several recent learning-based methods also use similar multi-layer image representations [7, 60]. However, these methods use “rigid” layer structures, in the sense that every pixel in the image has the same (fixed and predetermined) number of layers. At every pixel, they store the nearest surface in the first layer, the second-nearest in the next layer, etc. This is problematic, because across depth discontinuities the content within a layer changes abruptly, which destroys locality in receptive fields of convolution kernels.

In this work we present a new learning-based method that generates a 3D photo from an RGB-D input. The depth can either come from dual camera cell phone stereo, or be estimated from a single RGB image [31, 29, 13]. We use the LDI representation (similar to Facebook 3D Photos) because it is compact and allows us to handle situations of arbitrary depth-complexity. Unlike the “rigid” layer structures described above, we explicitly store connectivity across pixels in our representation. However, as a result it is more difficult to apply a *global* CNN to the problem, because our topology is more complex than a standard tensor. Instead, we break the problem into many local inpainting sub-problems, which we solve iteratively. Each problem is locally like an image, so we can apply standard CNN. We use an inpainting model that is conditioned on *spatially-adaptive* context regions, which are extracted from the local connectivity of the LDI. After synthesis we fuse the inpainted regions back into the LDI, leading to a recursive algorithm that proceeds until all depth edges are treated.

The result of our algorithm are 3D photos with synthesized texture and structures in occluded regions (Figure 1d). Unlike most previous approaches we do not require pre-determining a fixed number of layers. Instead our algorithm adapts by design to the local depth-complexity of the input and generates a varying number of layers across the image. We have validated our approach on a wide variety of photos captured in different situations.

2. Related Work

Representation for novel view synthesis. Different types of representations have been explored for novel view synthesis, including light fields [15, 30, 2], multi-plane images [77, 56, 40], and layered depth images [51, 59, 7, 60, 17, 18, 6, 44]. Light fields enable photorealistic rendering of novel views, but generally require many input images

to achieve good results. The multi-plane image representation [77, 56, 40] stores multiple layers of RGB- α images at fixed depths. The main advantage of this representation is its ability to capture semi-reflective or semi-transparent surfaces. However, due to the fixed depth discretization, sloped surfaces often do not reproduce well, unless an excessive number of planes is used. Many variants of layered depth image representations have been used over time. Representations with a fixed number of layers everywhere have recently been used [7, 60], but they do not preserve locality well, as described in the previous section. Other recent work [17, 18] extends the original work of Shade et al. [51] to explicitly store connectivity information. This representation can locally adapt to any depth-complexity and can be easily converted into a textured mesh for efficient rendering. Our work uses this representation as well.

Image-based rendering. Image-based rendering techniques enable photorealistic synthesis of novel views from a collection of posed images. These methods work best when the images have sufficiently large baselines (so that multi-view stereo algorithms can work well) or are captured with depth sensors. Recent advances include learning-based blending [19], soft 3D reconstruction [47], handling reflection [53, 27], relighting [68], and reconstructing mirror and glass surfaces [64]. Our focus in this work lies in novel view synthesis *from one single image*.

Learning-based view synthesis. CNN-based methods have been applied to synthesizing novel views from sparse light field data [24] or two or more posed images [12, 19, 4]. Several recent methods explore view synthesis from a single image. These methods, however, often focus on a specific domain [57, 65], synthetic 3D scenes/objects [78, 45, 58, 6, 7, 11], hallucinating only one specific view [66, 73], or assuming piecewise planar scenes [33, 35].

Many of these learning-based view synthesis methods require running a forward pass of the pre-trained network to synthesize the image of a given viewpoint. This makes these approaches less applicable to display on resource-constrained devices. Our representation, on the other hand, can be easily converted into a textured mesh and efficiently rendered with standard graphics engines.

Image inpainting. The task of image inpainting aims to fill missing regions in images with plausible content. Inspired by the success of texture synthesis [9, 8], *example-based methods* complete the missing regions by transferring the contents from the known regions of the image, either through non-parametric patch-based synthesis [63, 1, 5, 20] or solving a Markov Random Field model using belief propagation [26] or graph cut [48, 28, 16]. Driven by the progress of convolutional neural networks, *CNN-based methods* have received considerable attention due to their ability to predict semantically meaningful contents that are not available in the known regions [46, 55, 21, 70, 71]. Recent efforts include designing CNN architectures to better handle holes with irregular shapes [34, 72, 69] and two-

stage methods with structure-content disentanglement, e.g., predicting structure (e.g., contour/edges in the missing regions) and followed by content completion conditioned on the predicted structures [43, 67, 49].

Our inpainting model builds upon the recent two-stage approaches [43, 67, 49] but with two key differences. First, unlike existing image inpainting algorithms where the hole and the available contexts are *static* (e.g., the known regions in the entire input image), we apply the inpainting *locally* around each depth discontinuity with *adaptive* hole and context regions. Second, in addition to inpaint the color image, we also inpaint the depth values as well as the depth discontinuity in the missing regions.

Depth inpainting. Depth inpainting has applications in filling missing depth values where commodity-grade depth cameras fail (e.g., transparent/reflective/distant surfaces) [36, 75, 37] or performing image editing tasks such as object removal on stereo images [62, 42]. The goal of these algorithms, however, is to inpaint the depth of the *visible surfaces*. In contrast, our focus is on recovering the depth of the *hidden surface*.

CNN-based single depth estimation. CNN-based methods have recently demonstrated promising results on estimating depth from a single image. Due to the difficulty of collecting labeled datasets, earlier approaches often focus on specific visual domains such as indoor scenes [10] or street view [14, 76]. While the accuracy of these approaches is not yet competitive with multi-view stereo algorithms, this line of research is particularly promising due to the availability of larger and more diverse training datasets from relative depth annotations [3], multi-view stereo [31], 3D movies [29] and synthetic data [44].

For cases where only one single color image is available, we obtain the depth estimate through a pre-trained depth estimation model [31, 29]. Removing the dependency on stereo or multiple images as input makes our method more widely applicable to all the existing photos.

3. Method

Layered depth image. Our method takes as input an RGB-D image (i.e., an aligned color-and-depth image pair) and generates a *Layered Depth Image* (LDI, [51]) with inpainted color and depth in parts that were occluded in the input.

An LDI is similar to a regular 4-connected image, except at every position in the pixel lattice it can hold any number of pixels, from zero to many. Each LDI pixel stores a color and a depth value. Unlike the original LDI work [51], we explicitly represent the local connectivity of pixels: each pixel stores pointers to either zero or at most one direct neighbor in each of the four cardinal directions (left, right, top, bottom). LDI pixels are 4-connected like normal image pixels within smooth regions, but do not have neighbors across depth discontinuities.

LDIs are a useful representation for 3D photography, be-

cause (1) they naturally handle an arbitrary number of layers, i.e., can adapt to depth-complex situations as necessary, and (2) they are sparse, i.e., memory and storage efficient and can be converted into a light-weight textured mesh representation that renders fast.

The quality of the depth input to our method does not need to be perfect, as long as discontinuities are reasonably well aligned in the color and depth channels. In practice, we have successfully used our method with inputs from dual camera cell phones as well as with estimated depth maps from learning-based methods [31, 29].

Method overview. Given an input RGB-D image, our method proceeds as follows. We first initialize a trivial LDI, which uses a single layer everywhere and is fully 4-connected. In a pre-process we detect major depth discontinuities and group them into simple connected *depth edges* (Section 3.1). These form the basic units for our main algorithm below. In the core part of our algorithm, we iteratively select a depth edge for inpainting. We then *disconnect* the LDI pixels across the edge and only consider the background pixels of the edge for inpainting. We extract a local *context region* from the “known” side of the edge, and generate a *synthesis region* on the “unknown” side (Section 3.2). The synthesis region is a contiguous 2D region of *new* pixels, whose color and depth values we generate from the given context using a learning-based method (Section 3.3). Once inpainted, we merge the synthesized pixels back into the LDI (Section 3.4). Our method iteratively proceeds in this manner until all depth edges have been treated.

3.1. Image preprocessing

The only input to our method is a single RGB-D image. Every step of the algorithm below proceeds fully automatically. We normalize the depth channel, by mapping the min and max disparity values (i.e., 1 / depth) to 0 and 1, respectively. All parameters related to spatial dimensions below are tuned for images with 1024 pixels along the longer dimension, and should be adjusted proportionally for images of different sizes.

We start by lifting the image onto an LDI, i.e., creating a single layer everywhere and connecting every LDI pixel to its four cardinal neighbors. Since our goal is to inpaint the occluded parts of the scene, we need to find *depth discontinuities* since these are the places where we need to extend the existing content. In most depth maps produced by stereo methods (dual camera cell phones) or depth estimation networks, discontinuities are blurred across multiple pixels (Figure 2c), making it difficult to precisely localize them. We, therefore, sharpen the depth maps using a bilateral median filter [38] (Figure 2d), using a 7×7 window size, and $\sigma_{spatial} = 4.0$, $\sigma_{intensity} = 0.5$.

After sharpening the depth map, we find discontinuities by thresholding the disparity difference between neighboring pixels. This results in many spurious responses, such as isolated speckles and short segments dangling off longer

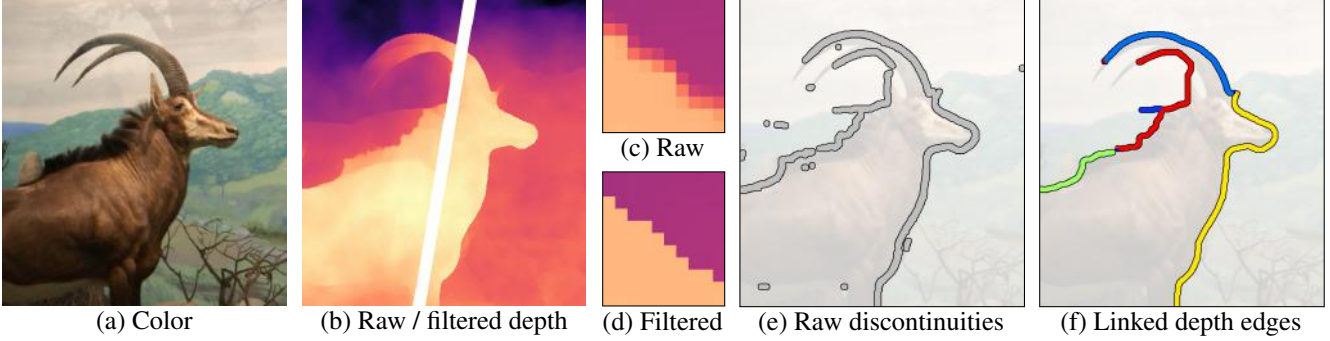


Figure 2. **Preprocessing.** Preprocessing of the color and depth input (a-b). We use a bilateral median filter to sharpen the input depth maps (c-d), detect raw discontinuities using disparity thresholds (e), and clean up spurious threshold responses and link discontinuities into connected depth edges (f). These linked depth edges form the basic unit for our inpainting process.

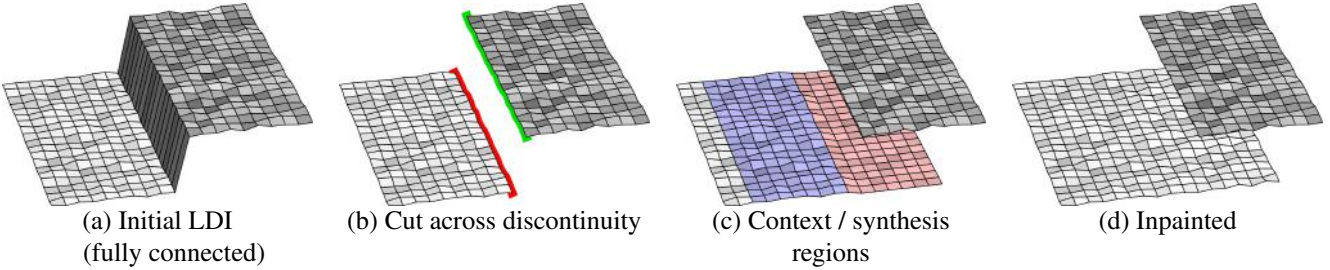


Figure 3. **Conceptual illustration of the LDI inpainting algorithm.** (a) The initial LDI is fully connected. A depth edge (discontinuity) is marked in gray. (b) We first cut the LDI pixel connections across the depth, forming a foreground silhouette (green) and a background silhouette (red). (c) For the background silhouette we spawn a context region (blue) and a synthesis region (red) of new LDI pixels. (d) The synthesized pixels have been merged into the LDI.



Figure 4. **Context/synthesis regions.** Context regions (blue) and synthesis regions (red) for three example connected depth edges (black) from Figure 2(f).

edges (Figure 2e). We clean this up as follows: First, we create a binary map by labeling depth discontinuities as 1 (and others as 0). Next, we use connected component analysis to merge adjacent discontinuities into a collection of “linked depth edges”. To avoid merging edges at junctions, we separate them based on the local connectivity of the LDI. Finally, we remove short segments (< 10 pixels), including both isolated and dangling ones. We determine the threshold 10 by conducting five-fold cross-validation with LPIPS [74] metric on 50 samples randomly selected from RealEstate10K training set. The final edges (Figures 2f) form the basic unit of our iterative inpainting procedure, which is described in the following sections.

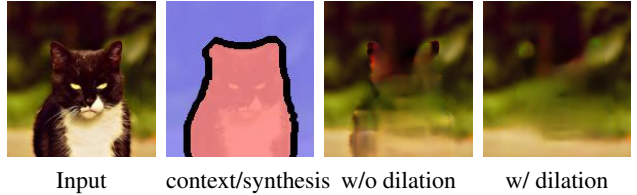


Figure 5. **Handling imperfect depth edges.** As the detected depth edges may not align well around occlusion boundaries, we dilate the synthesis region by 5 pixels. This strategy helps reduce artifacts in the inpainted regions.

3.2. Context and synthesis regions

Our inpainting algorithm operates on one of the previously computed depth edges at a time. Given one of these edges (Figure 3a), the goal is to synthesize new color and depth content in the adjacent occluded region. We start by disconnecting the LDI pixels across the discontinuity (Figure 3b). We call the pixels that became disconnected (i.e., are now missing a neighbor) *silhouette* pixels. We see in Figure 3b that a foreground silhouette (marked green) and a background silhouette (marked red) forms. Only the background silhouette requires inpainting. We are interested in extending its surrounding content into the occluded region.

We start by generating a *synthesis region*, a contiguous region of *new* pixels (Figure 3c, red pixels). These are essentially just 2D pixel coordinates at this point. We initialize the color and depth values in the synthesis region

using a simple iterative flood-fill like algorithm. It starts by stepping from all silhouette pixels one step in the direction where they are disconnected. These pixels form the initial synthesis region. We then iteratively expand (for 40 iterations) all pixels of the region by stepping left/right/up/down and adding any pixels that have not been visited before. For each iteration, we expand the context and synthesis regions alternately and thus a pixel only belong to either one of the two regions. Additionally, we do not step back *across* the silhouette, so the synthesis region remains strictly in the occluded part of the image. Figure 4 shows a few examples.

We describe our learning-based technique for inpainting the synthesis region in the next section. Similar techniques [34, 43] were previously used for filling holes in images. One important difference to our work is that these image holes were always fully surrounded by known content, which constrained the synthesis. In our case, however, the inpainting is performed on a connected layer of an LDI pixels, and it should only be constrained by surrounding pixels that are directly connected to it. Any other region in the LDI, for example on other foreground or background layer, is entirely irrelevant for this synthesis unit, and should not constrain or influence it in any way.

We achieve this behavior by explicitly defining a *context region* (Figure 3c, blue region) for the synthesis. Our inpainting networks only considers the content in the context region and does not see any other parts of the LDI. The context region is generated using a similar flood-fill like algorithm. One difference, however, is that this algorithm selects actual LDI pixels and follows their connection links, so the context region expansion halts at silhouettes. We run this algorithm for 100 iterations, as we found that synthesis performs better with slightly larger context regions. In practice, the silhouette pixels may not align well with the actual occluding boundaries due to imperfect depth estimation. To tackle this issue, we dilate the synthesis region near the depth edge by 5 pixels (the context region erodes correspondingly). Figure 5 shows the effect of this heuristic.

3.3. Context-aware color and depth inpainting

Model. Given the context and synthesis regions, our next goal is to synthesize color and depth values. Even though we perform the synthesis on an LDI, the extracted context and synthesis regions are locally like images, so we can use standard network architectures designed for images. Specifically, we build our color and depth inpainting models upon image inpainting methods in [43, 34, 67].

One straightforward approach is to inpaint the color image and depth map independently. The inpainted depth map, however, may not be well-aligned with respect to the inpainted color. To address this issue, we design our color and depth inpainting network similar to [43, 67]: we break down the inpainting tasks into three sub-networks: (1) edge inpainting network, (2) color inpainting network, and (3) depth inpainting network (Figure 6). First, given the con-

text edges as input, we use the edge inpainting network to predict the depth edges in the synthesis regions, producing the inpainted edges. Performing this step first helps infer the *structure* (in terms of depth edges) that can be used for constraining the *content* prediction (the color and depth values). We take the concatenated inpainted edges and context color as input and use the color inpainting network to produce inpainted color. We perform the depth inpainting similarly. Figure 7 shows an example of how the edge-guided inpainting is able to extend the depth structures accurately and alleviate the color/depth misalignment issue.

Multi-layer inpainting. In depth-complex scenarios, applying our inpainting model once is not sufficient as we can still see the hole through the discontinuity created by the *inpainted* depth edges. We thus apply our inpainting model until no further inpainted depth edges are generated. Figure 8 shows an example of the effects. Here, applying our inpainting model once fills in missing layers. However, several holes are still visible when viewed at a certain viewpoint (Figure 8b). Applying the inpainting model one more time fixes the artifacts.

Training data generation. For training, our proposed model can be simply trained on any image dataset without the need of annotated data. Here, we choose to use MSCOCO dataset [32] for its wide diversity in object types and scenes. To generate the training data for the inpainting model, we create a synthetic dataset as follows. First, we apply the pre-trained MegaDepth [31] on the COCO dataset to obtain pseudo ground truth depth maps. We extract context/synthesis regions (as described in Section 3.2) to form a pool of these regions. We then randomly sample and place these context-synthesis regions on *different* images in the COCO dataset. We thus can obtain the ground truth content (RGB-D) from the *simulated* occluded region.

3.4. Converting to 3D textured mesh

We form the 3D textured mesh by integrating all the inpainted depth and color values back into the original LDI. Using mesh representations for rendering allows us to quickly render novel views, without the need to perform per-view inference step. Consequently, the 3D representation produced by our algorithm can easily be rendered using standard graphics engines on edge devices.

4. Experimental Results

In this section, we start with describing implementation details (Section 4.1). We then show visual comparisons with the state-of-the-art novel view synthesis methods (Section 4.2). We refer to the readers to supplementary material for extensive results and comparisons. Next, we follow the evaluation protocol in [77] and report the quantitative comparisons on the RealEstate10K dataset (Section 4.3). We present an ablation study to justify our model design (Section 4.4). Finally, we show that our method works well with

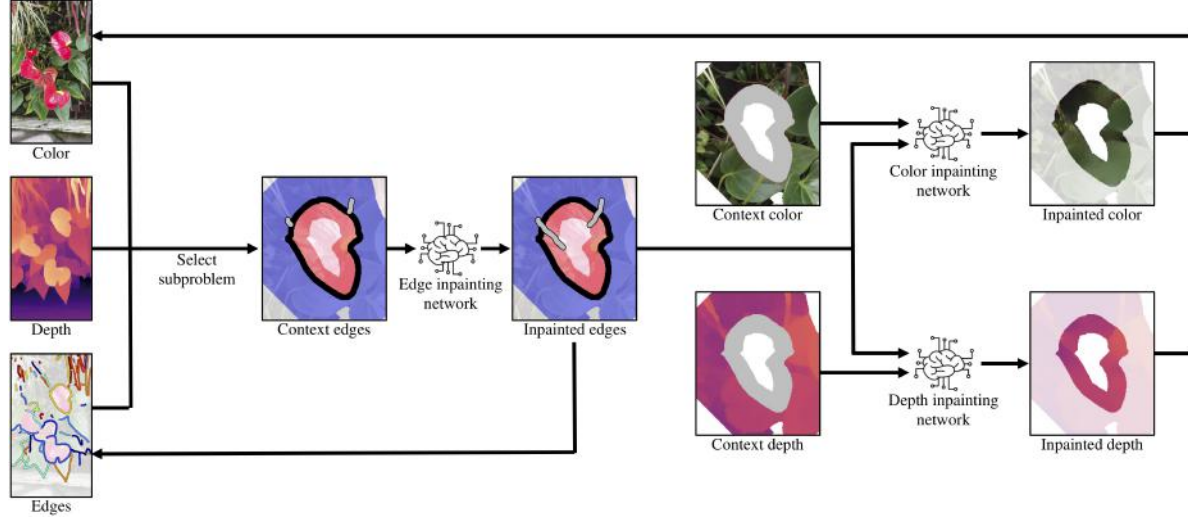


Figure 6. **Context-aware color and depth inpainting.** Given the color, depth, the extracted and linked depth edges as inputs, we randomly select one of the edges as a subproblem. We start with inpainting the depth edge in the synthesis region (red) using an *edge inpainting network*. We then concatenate the inpainted depth edges with the context color together and apply a *color inpainting network* to produce the inpainted color. Similarly, we concatenate the inpainted depth edges with the context depth and apply a *depth inpainting network* to produce the inpainted depth.

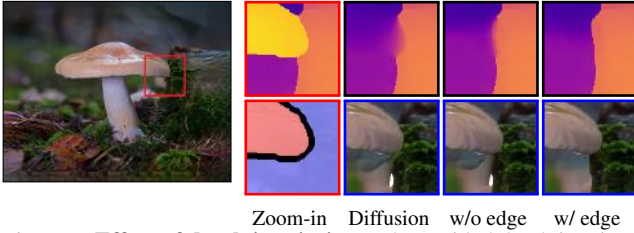


Figure 7. **Effect of depth inpainting.** Edge-guided depth inpainting produces more accurate structure inpainting, particularly for depth-complex regions (e.g., T-junctions). **Blue box:** synthesized novel view.

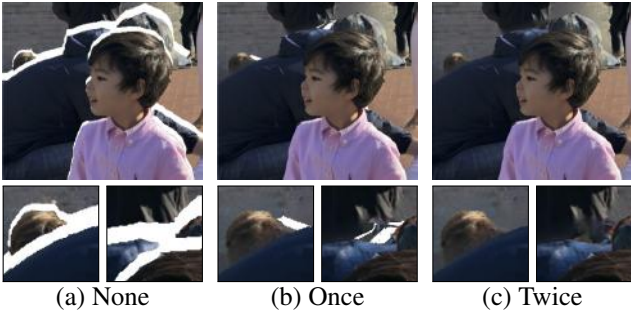


Figure 8. **Multi-layer inpainting.**

depth maps from different sources (Section 4.5). Additional details and visual comparisons can be found in our supplementary material.

4.1. Implementation details

Training the inpainting model. For the edge-generator, we follow the hyper-parameters in [43]. Specifically, we train the edge-generator model using the ADAM optimizer [25] with $\beta = 0.9$ and an initial learning rate of

0.0001. We train both the edge and depth generator model using the context-synthesis regions dataset on the MS-COCO dataset for 5 epochs. We train the depth generator and color image generator for 5 and 10 epochs, respectively.

Inpainting model architecture. For the edge inpainting network, we adopt the architecture provided by [43]. For the depth and color inpainting networks, we use a standard U-Net architecture with partial convolution [34]. Due to the space limitation, we leave additional implementation details (specific network architecture, the training loss and the weights for each network) to the supplementary material. We will make the source code and pre-trained model publicly available to foster future work.

Training data. We use the 118k images from COCO 2017 set for training. We select at most 3 pairs of regions from each image to form the context-synthesis pool. During training, we sample one pair of regions for each image, and resize it by a factor between [1.0, 1.3].

4.2. Visual comparisons

Comparisons with methods with MPI representations. We compare our proposed model against MPI-based approaches on RealEstate10K dataset. We use DPSNet [22] to obtain the input depth maps for our method. We render the novel views of MPI-based methods using the pre-trained weights provided by the authors. Figure 9 shows two challenging examples with complex depth structures. Our method synthesizes plausible structures around depth boundaries; on the other hand, stereo magnification and PB-MPI produce artifacts around depth discontinuities. LLFF [39] suffers from ghosting effects when extrapolating new views.

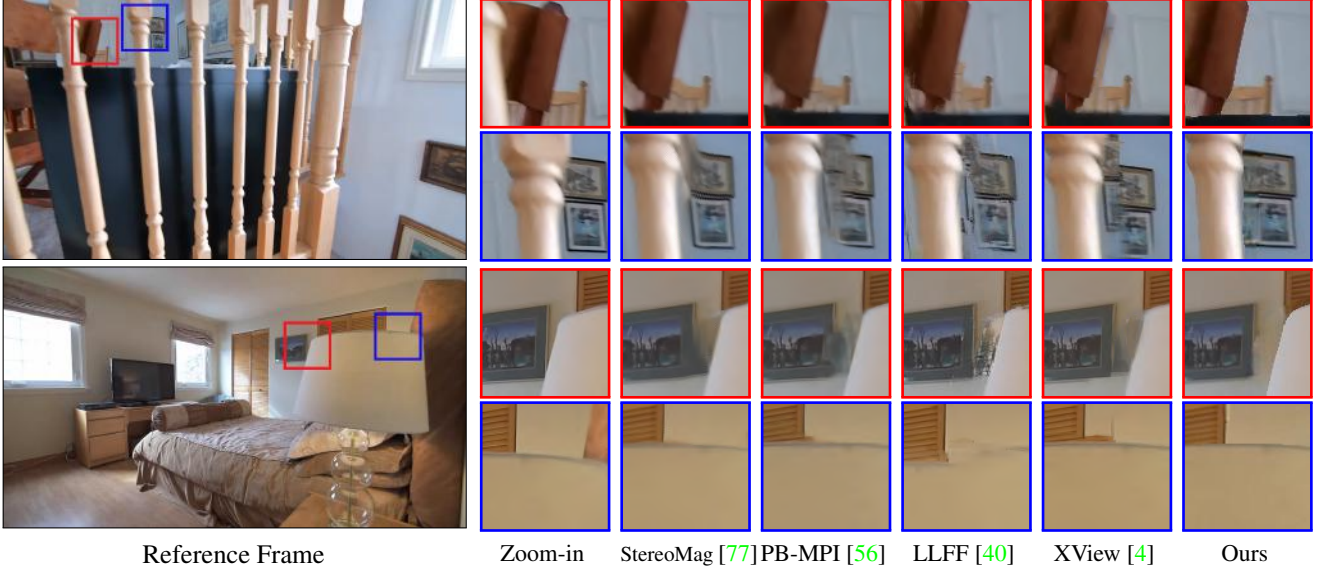


Figure 9. **Visual comparison with MPI-based methods.** Our method inpaints plausible structure and color in the occluded region.

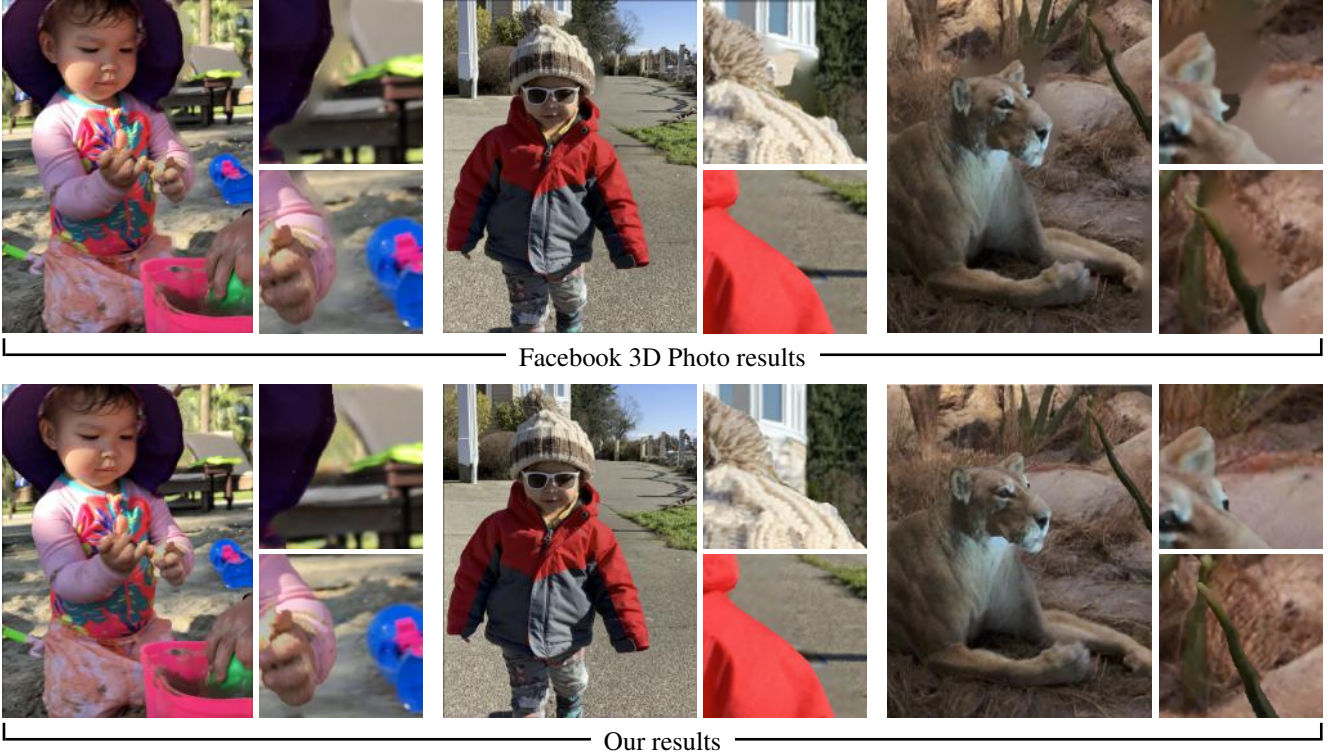


Figure 10. **Visual comparison to Facebook 3D Photos.** Our approach fills plausible textures and structures at disocclusions.

Comparisons with Facebook 3D photo. Here, we aim to evaluate the capability of our method on photos taken *in the wild*. We extract the color images and the corresponding depth maps estimated from an iPhone X (with dual camera lens). We use the same set of RGB-D inputs for both Facebook 3D photo and our algorithm. Figure 10 shows the view synthesis result in comparison with Facebook 3D photo. The diffused color and depth values by the facebook

3D photo algorithm work well when small or thin occluded regions are revealed at novel views. These artifacts, however, become clearly visible with larger occluded regions. On the other hand, our results in general fills in the synthesis regions with visually plausible contents and structures.

Table 1. **Quantitative comparison** on the RealEstate10K dataset.

Methods	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow
Stereo-Mag [77]	0.8906	26.71	0.0826
PB-MPI [56]	0.8773	25.51	0.0902
LLFF [40]	0.8697	24.15	0.0941
Xview [4]	0.8628	24.75	0.0822
Ours	0.8887	27.29	0.0724

Table 2. **Using depth edge as guidance improves the results.** Blue: results in disoccluded regions.

Methods	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow
Diffusion	0.8665 (0.6237)	25.95 (18.91)	0.084
Inpaint w/o edge	0.8665 (0.6247)	25.96 (18.94)	0.084
Inpaint w/ edge (Ours)	0.8666 (0.6265)	25.97 (18.98)	0.083

Table 3. **Using color inpainting model gives better perceptual quality.** Our dilation heuristic further boosts the performance. Blue: results in disoccluded regions.

Methods	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow
Diffusion	0.8661 (0.6215)	25.90 (18.78)	0.088
Inpaint w/o dilation	0.8643 (0.5573)	25.56 (17.14)	0.085
Inpaint w/ dilation (Ours)	0.8666 (0.6265)	25.97 (18.98)	0.083

4.3. Quantitative comparisons

We evaluate how well our model can extrapolate views compared to MPI-based methods [56, 77, 4, 40]. We randomly sample 1500 video sequences from RealEstate10K to generate testing triplets. For each triplet, we set $t = 10$ for target view, so that all the methods need to extrapolate beyond the source ($t = 0$) and reference ($t = 4$) frame. We use DPSNet [22] to generate the input depth maps required for our model. We quantify the performance of each model using SSIM and PSNR metrics between the synthesized target views and the ground truth. As these metrics do not capture the perceptual quality of the synthesized view, we include LPIPS [74] metric to quantify how well does the generated view align with human perception. For PB-MPI, we set the number of depth layers to 64 as it yields the best result. We report the evaluation results in Table 1. Our proposed method performs competitively on SSIM and PSNR. In addition, our synthesis views exhibit better perceptual quality, as reflected in the superior LPIPS score.

4.4. Ablation study

We conduct ablation studies to see how each of our proposed components contribute to the final performance. We first verify the effectiveness of *edge-guided depth inpainting*. We sample 130 triplets from our testing sequences, evaluate the inpainted color on both the entire image and disoccluded regions, and report the numbers in Table 2. The results show that our proposed edge-guided inpainting leads to minor improvement in numerical metrics. Next, we ex-

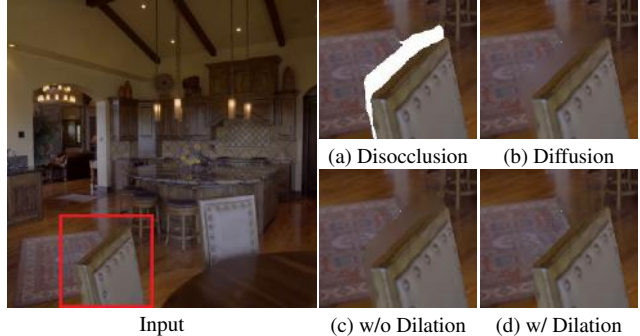


Figure 11. **Color inpainting leads to better visual quality.**

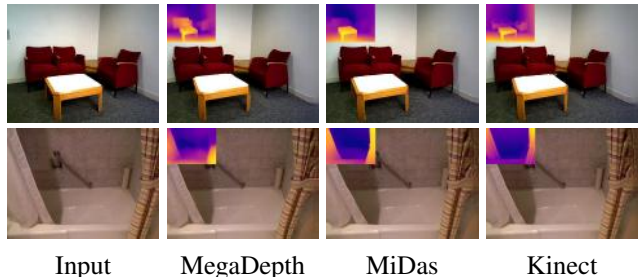


Figure 12. **Our method works with various sources of depth map.** We show the depth estimates on the top-left of novel views.

amine the efficacy of our *color inpainting* model following the same procedure described above. We present the performance in both entire image and occluded regions in Table 3. We observe that our proposed model yields better perceptual quality. Figure 11 shows an example.

4.5. Handling different depth maps

We test our method using depth maps generated using different approaches (Figure 12). We select images from SUNRGBD [54] dataset, and obtain the corresponding depth maps from three different sources: 1) depth estimated with MegaDepth [31], 2) MiDas [29] and 3) Kinect depth sensor. We present the resulting 3D photos in Figure 12. The results show that our method can handle depth maps from different sources reasonably well.

5. Conclusions

In this paper, we present an algorithm for creating compelling 3D photography from a single RGB-D image. Our core technical novelty lies in creating a completed layered depth image representation through context-aware color and depth inpainting. We validate our method on a wide variety of everyday scenes. Our experimental results show that our algorithm produces considerably fewer visual artifacts when compared with the state-of-the-art novel view synthesis techniques. We believe that such technology can bring 3D photography to a broader community, allowing people to easily capture scenes for immersive viewing.

Acknowledgement. This project is supported in part by NSF (#1755785) and MOST-108-2634-F-007-006 and MOST-109-2634-F-007-016.

References

- [1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. In *ACM Transactions on Graphics*, volume 28, page 24, 2009. 2
- [2] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001. 2
- [3] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. In *NeurIPS*, 2016. 3
- [4] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. In *ICCV*, 2019. 2, 7, 8, 12
- [5] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics*, 31(4):82–1, 2012. 2
- [6] Helisa Dhama, Nassir Navab, and Federico Tombari. Object-driven multi-layer scene decomposition from a single image. In *ICCV*, 2019. 2
- [7] Helisa Dhama, Keisuke Tateno, Iro Laina, Nassir Navab, and Federico Tombari. Peeking behind objects: Layered depth prediction from a single image. In *ECCV*, 2018. 2
- [8] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, 2001. 2
- [9] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999. 2
- [10] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 3
- [11] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 2
- [12] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world’s imagery. In *CVPR*, 2016. 1, 2
- [13] Clément Godard, Oisín Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *ICCV*, pages 3828–3838, 2019. 2
- [14] Clément Godard, Oisín Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017. 3
- [15] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *SIGGRAPH*, volume 96, pages 43–54, 1996. 2
- [16] Kaiming He and Jian Sun. Image completion approaches using the statistics of similar patches. *TPAMI*, 36(12):2423–2435, 2014. 2
- [17] Peter Hedman, Suhb Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3d photography. *ACM Transactions on Graphics*, 36(6):234, 2017. 1, 2
- [18] Peter Hedman and Johannes Kopf. Instant 3d photography. *ACM Transactions on Graphics*, 37(4):101, 2018. 1, 2
- [19] 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*, page 257, 2018. 1, 2
- [20] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. Image completion using planar structure guidance. *ACM Transactions on graphics*, 33(4):129, 2014. 2
- [21] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *TOG*, 36(4):107, 2017. 2
- [22] Sunghoon Im, Hae-Gon Jeon, Steve Lin, and In So Kweon. Dpsnet: End-to-end deep plane sweep stereo. 2019. 6, 8
- [23] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 12
- [24] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Transactions on Graphics*, 35(6):193, 2016. 2
- [25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6
- [26] Nikos Komodakis and Georgios Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *TIP*, 16(11):2649–2661, 2007. 2
- [27] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. Image-based rendering in the gradient domain. *ACM Transactions on Graphics*, 32(6):199, 2013. 1, 2
- [28] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003. 2
- [29] 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. 2, 3, 8
- [30] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996. 2
- [31] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *CVPR*, 2018. 2, 3, 5, 8, 14
- [32] 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 *ECCV*, 2014. 5
- [33] Chen Liu, Jimei Yang, Duygu Ceylan, Ersin Yumer, and Yasutaka Furukawa. Planenet: Piece-wise planar reconstruction from a single rgb image. In *CVPR*, 2018. 2
- [34] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *ECCV*, 2018. 2, 5, 6, 12
- [35] Miaomiao Liu, Xuming He, and Mathieu Salzmann. Geometry-aware deep network for single-image novel view synthesis. In *CVPR*, 2018. 2
- [36] Wei Liu, Xiaogang Chen, Jie Yang, and Qiang Wu. Robust color guided depth map restoration. *TIP*, 26(1):315–327, 2017. 3

- [37] Si Lu, Xiaofeng Ren, and Feng Liu. Depth enhancement via low-rank matrix completion. In *CVPR*, 2014. 3
- [38] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. *Proceedings of the 2013 IEEE International Conference on Computer Vision*, pages 49–56, 2013. 3
- [39] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM, 1995. 6
- [40] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4), July 2019. 1, 2, 7, 8, 12
- [41] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. 2018. 14
- [42] Tai-Jiang Mu, Ju-Hong Wang, Song-Pei Du, and Shi-Min Hu. Stereoscopic image completion and depth recovery. *The Visual Computer*, 30(6-8):833–843, 2014. 3
- [43] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *arXiv preprint*, 2019. 3, 5, 6, 12, 14
- [44] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Transactions on Graphics (TOG)*, 38(6), Nov. 2019. 2, 3
- [45] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C Berg. Transformation-grounded image generation network for novel 3d view synthesis. In *CVPR*, 2017. 2
- [46] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 2
- [47] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):235, 2017. 1, 2
- [48] Yael Pritch, Eitam Kav-Venaki, and Shmuel Peleg. Shift-map image editing. In *CVPR*, pages 151–158. IEEE, 2009. 2
- [49] Yurui Ren, Xiaoming Yu, Ruonan Zhang, Thomas H Li, Shan Liu, and Ge Li. Structureflow: Image inpainting via structure-aware appearance flow. In *ICCV*, 2019. 3
- [50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 12
- [51] 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. ACM, 1998. 2, 3
- [52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 12
- [53] Sudipta N Sinha, Johannes Kopf, Michael Goesele, Daniel Scharstein, and Richard Szeliski. Image-based rendering for scenes with reflections. *ACM Transactions on Graphics*, 31(4):100–1, 2012. 2
- [54] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 8
- [55] Yuhang Song, Chao Yang, Zhe Lin, Xiaofeng Liu, Qin Huang, Hao Li, and C-C Jay Kuo. Contextual-based image inpainting: Infer, match, and translate. *arXiv preprint arXiv:1711.08590*, 2017. 2
- [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 *CVPR*, 2019. 1, 2, 7, 8, 12
- [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 *ICCV*, 2017. 2
- [58] 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 *ECCV*, 2018. 2
- [59] Lech Świrski, Christian Richardt, and Neil A Dodgson. Layered photo pop-up. In *ACM SIGGRAPH 2011 Posters*, 2011. 2
- [60] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3d scene inference via view synthesis. In *ECCV*, 2018. 2
- [61] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 14
- [62] Liang Wang, Hailin Jin, Ruigang Yang, and Minglun Gong. Stereoscopic inpainting: Joint color and depth completion from stereo images. In *CVPR*, 2008. 3
- [63] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *TPAMI*, (3):463–476, 2007. 2
- [64] Thomas Whelan, Michael Goesele, Steven J Lovegrove, Julian Straub, Simon Green, Richard Szeliski, Steven Butterfield, Shobhit Verma, and Richard Newcombe. Reconstructing scenes with mirror and glass surfaces. *ACM Transactions on Graphics*, 37(4):102, 2018. 1, 2
- [65] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *CVPR*, 2020. 2
- [66] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *ECCV*, 2016. 2
- [67] Wei Xiong, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-aware image inpainting. 2019. 3, 5
- [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)*, 38(4):76, 2019. 2
- [69] Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-net: Image inpainting via deep feature rearrangement. In *ECCV*, September 2018. 2
- [70] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *CVPR*, volume 1, page 3, 2017. 2
- [71] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018. 2

- [72] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. [2](#)
- [73] Qiong Zeng, Wenzheng Chen, Huan Wang, Changhe Tu, Daniel Cohen-Or, Dani Lischinski, and Baoquan Chen. Hallucinating stereoscopy from a single image. In *Computer Graphics Forum*, volume 34, pages 1–12, 2015. [2](#)
- [74] 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. [4](#), [8](#)
- [75] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single rgb-d image. In *CVPR*, 2018. [3](#)
- [76] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, volume 2, page 7, 2017. [3](#)
- [77] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Transactions on Graphics*, 2018. [1](#), [2](#), [5](#), [7](#), [8](#), [12](#)
- [78] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *ECCV*, 2016. [2](#)

3D Photography using Context-aware Layered Depth Inpainting

Supplementary Material

Table 4. **Quantitative comparison** on the RealEstate10K dataset.

Methods	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow
Stereo-Mag [77]	0.8906	26.71	0.0826
PB-MPI [56] (32 Layers)	0.8717	25.38	0.0925
PB-MPI [56] (64 Layers)	0.8773	25.51	0.0902
PB-MPI [56] (128 Layers)	0.8700	24.95	0.1030
LLFF [40]	0.8697	24.15	0.0941
Xview [4]	0.8628	24.75	0.0822
Ours	0.8887	27.29	0.0724

6. Additional Quantitative Results

We further evaluate the PB-MPI method [56] with various number of depth layers. We report the results in Table 4.

7. Visual Results

Comparisons with the state-of-the-arts. We provide a collection of rendered 3D photos with comparisons with the state-of-the-art novel view synthesis algorithms. In addition, we show that our method can synthesize novel view for legacy photos. Please refer to the website³ for viewing the results.

Ablation studies. To showcase how each of our proposed component contribute to the quality of the synthesized view, we include a set of rendered 3D photos using the same ablation settings in Section 4.4 of the main paper. Please refer to the website³ for viewing the photos.

8. Implementation Details

In this section, we provide additional implementation details of our model, including model architectures, training objectives, and training dataset collection. We will release the source code to facilitate future research in this area.

Model architectures. We adopt the same U-Net [50] architecture as in [34] for our depth inpainting and color inpainting models (see Table 5), and change the input channels for each model accordingly. For the edge inpainting model, we use a design similar to [43] (see Table 6). We set the input depth and RGB values in the synthesis region to zeros for all three models. The input edge values in the synthesis region are similarly set to zeros for depth and color inpainting models, but remain intact for the edge inpainting network. We show the input details of each model in Table 7

Training objective. To train our color inpainting model, we adopt similar objective functions as in [34]. First, we define the reconstruction loss for context and synthesis regions:

$$L_{\text{synthesis}} = \frac{1}{N} \|S \odot (I - I_{gt})\|, \quad L_{\text{context}} = \frac{1}{N} \|C \odot (I - I_{gt})\|, \quad (1)$$

where S and C are the binary mask indicating *synthesis* and *context* regions, respectively, \odot denotes the Hadamard product, N is the total number of pixels, I is the inpainted result, and I_{gt} is the ground truth image.

Next, we define the perceptual loss [23]:

$$L_{\text{perceptual}} = \sum_p^{P-1} \frac{\|\psi_p(I) - \psi_p(I_{gt})\|}{N_{\psi_p}}, \quad (2)$$

Here, $\psi_p(\cdot)$ is the output of the p th layer from VGG-16 [52], and N_{ψ_p} is the total number of elements in $\psi_p(\cdot)$.

³<https://shihmengli.github.io/3D-Photo-Inpainting/>

Table 5. **Model architecture of our color and depth inpainting models.** W denote partial convolution layer as PConv, and denote BatchNorm as BN. We add the context and synthesis region together as the partial masks for the PConv layers.

Module	Filter Size	#Channels	Dilation	Stride	Norm	Nonlinearity
PConv1	7×7	64	1	2	-	ReLU
PConv2	5×5	128	1	2	BN	ReLU
PConv3	5×5	256	1	2	BN	ReLU
PConv4	3×3	512	1	2	BN	ReLU
PConv5	3×3	512	1	2	BN	ReLU
PConv6	3×3	512	1	2	BN	ReLU
PConv7	3×3	512	1	2	BN	ReLU
PConv8	3×3	512	1	2	BN	ReLU
NearestUpsample	-	512	-	2	-	-
Concatenate (w/ PConv7)	-	512+512	-	-	-	-
PConv9	3×3	512	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	512	2	-	-	-
Concatenate (w/ PConv6)	-	512+512	-	-	-	-
PConv10	3×3	512	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	512	-	2	-	-
Concatenate (w/ PConv5)	-	512+512	1	-	-	-
PConv11	3×3	512	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	512	-	2	-	-
Concatenate (w/ PConv4)	-	512+512	-	-	-	-
PConv12	3×3	512	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	512	-	2	-	-
Concatenate (w/ PConv3)	-	512+256	-	-	-	-
PConv13	3×3	256	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	256	-	2	-	-
Concatenate (w/ PConv2)	-	256+128	-	-	-	-
PConv14	3×3	128	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	128	-	2	-	-
Concatenate (w/ PConv1)	-	128+64	-	-	-	-
PConv15	3×3	64	1	1	BN	LeakyReLU(0.2)
NearestUpsample	-	64	-	2	-	-
Concatenate (w/ Input)	-	64 + 4 or 64 + 6 (Depth / Color Inpainting)	-	-	-	-
PConv16	3×3	1 or 3 (Depth / Color Inpainting)	1	1	-	-

We define the style loss as:

$$L_{style} = \sum_p^{P-1} \frac{1}{C_p C_p} \left\| \frac{1}{C_p H_p W_p} \left[(\psi_p^I)^\top \psi_p^I - (\psi_p^{I_{gt}})^\top \psi_p^{I_{gt}} \right] \right\|, \quad (3)$$

where C_p, H_p, W_p is the number of channels, height, and width of the output $\psi_p(\cdot)$.

Finally, we adopt the Total Variation (TV) loss:

$$L_{tv} = \sum_{(i,j) \in S, (i,j+1) \in S} \frac{\|I(i,j+1) - I(i,j)\|}{N} + \sum_{(i,j) \in S, (i+1,j) \in S} \frac{\|I(i+1,j) - I(i,j)\|}{N}. \quad (4)$$

Here, We overload the notation S to denote the synthesis region. This term can be interpreted as a smoothing penalty on the synthesis area. Combine all these loss terms, we obtain the training objective for our color inpainting model:

$$L = L_{context} + 6L_{synthesis} + 0.05L_{perceptual} + 120L_{style} + 0.01L_{tv}$$

Table 6. **Model architecture of our edge inpainting models.** As in [43], the edge inpainting model consists of 1 edge generator, and 1 discriminator network. SN→IN indicates that we first perform spectral normalization (SN) [41], and then apply instance normalization (IN) [61]. ResnetBlock comprises 2 conv layers with the specified hyper-parameters and a skip connection between the input and the output of the block.

Edge Generator						
Module	Filter Size	#Channels	Dilation	Stride	Norm	Nonlinearity
Conv1	7×7	64	1	1	SN→IN	ReLU
Conv2	4×4	128	1	2	SN→IN	ReLU
Conv3	4×4	256	1	2	SN→IN	ReLU
ResnetBlock4	3×3	256	2	1	SN→IN	ReLU
ResnetBlock5	3×3	256	2	1	SN→IN	ReLU
ResnetBlock6	3×3	256	2	1	SN→IN	ReLU
ResnetBlock7	3×3	256	2	1	SN→IN	ReLU
ResnetBlock8	3×3	256	2	1	SN→IN	ReLU
ResnetBlock9	3×3	256	2	1	SN→IN	ReLU
ResnetBlock10	3×3	256	2	1	SN→IN	ReLU
ResnetBlock11	3×3	256	2	1	SN→IN	ReLU
ConvTranspose12	4×4	128	1	2	SN→IN	ReLU
ConvTranspose13	4×4	64	1	2	SN→IN	ReLU
Conv14	7×7	1	1	1	SN→IN	Sigmoid

Discriminator						
Module	Filter Size	#Channels	Dilation	Stride	Norm	Nonlinearity
Conv1	4×4	64	1	2	SN	LeakyReLU(0.2)
Conv2	4×4	128	1	2	SN	LeakyReLU(0.2)
Conv3	4×4	256	1	2	SN	LeakyReLU(0.2)
Conv4	4×4	512	1	1	SN	LeakyReLU(0.2)
Conv5	4×4	1	1	1	SN	Sigmoid

Table 7. **Input of each model in our proposed method.** The check mark ✓ indicates that it is used as input for the model.

	RGB	Depth	Edge	Context& Synthesis
Color Inpainting	✓	-	✓	✓
Depth Inpainting	-	✓	✓	✓
Edge Inpainting	✓	✓	✓	✓

For our depth inpainting model, we use only $L_{context} + L_{synthesis}$ as the objective functions. For edge inpainting model, we follow the identical training protocol as in [43].

Training details. We illustrate the data generation process in Figure 13. We use the depth map predicted by MegaDepth [31] as our pseudo ground truth. We train our method using 1 Nvidia V100 GPU with batch size of 8, and the total training time take about 5 days.

9. Failure cases

As estimating depth/disparity map from a single image remain a challenging problem (particularly for scenes with complex, thin structures), our method fails to produce satisfactory results with plausible motion parallax for scenes with complex structures. Due to the use of explicit depth map, our method is unable to handle reflective/transparent surfaces well. We show in Figure 14 two examples of such cases. Here, we show the input RGB image as well as the estimated depth map from the pre-trained MegaDepth model. The rendered 3D photos can be found in the supplementary webpage.

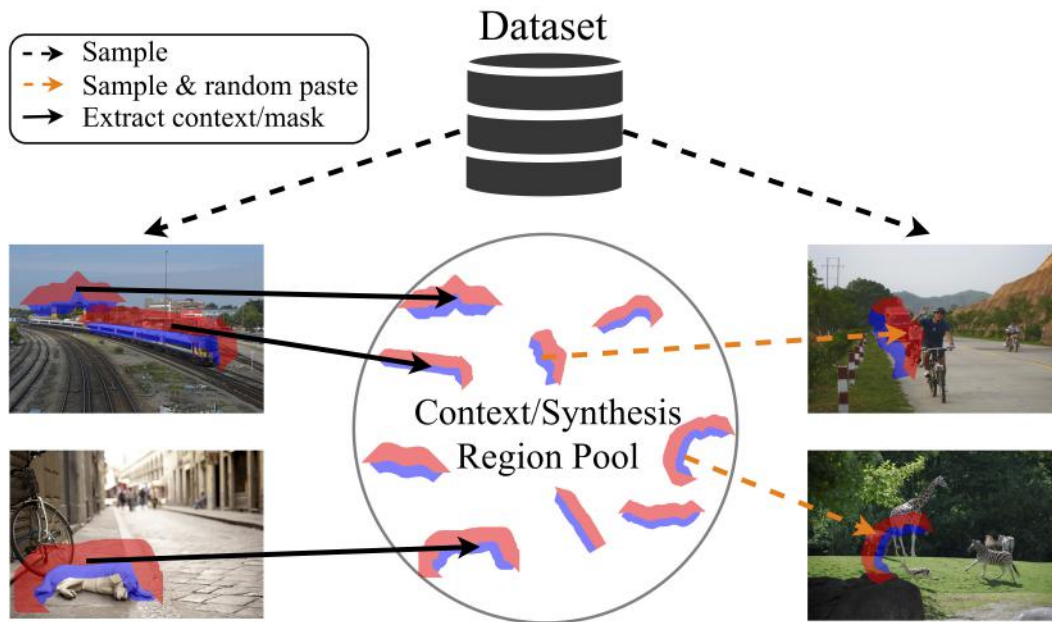


Figure 13. **Dataset generation process.** We first form a collection of context/synthesis regions by extracting them from the linked depth edges in images on the COCO dataset. We then randomly sample and paste these regions onto *different* images, forming our training dataset for context-aware color and depth inpainting.

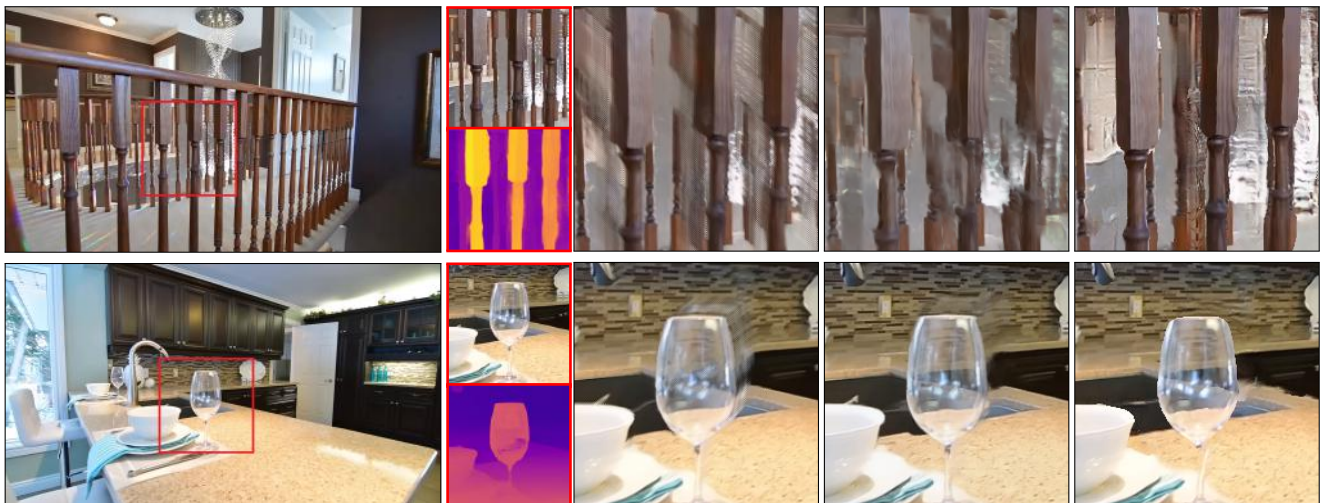


Figure 14. **Failure cases.** Single-image depth estimation algorithms (e.g., MegaDepth) often have difficulty in handling thin and complex structures and may produce overly smooth depth maps.