# Stereo Magnification: Learning view synthesis using multiplane images

TINGHUI ZHOU, University of California, Berkeley
RICHARD TUCKER, Google
JOHN FLYNN, Google
GRAHAM FYFFE, Google
NOAH SNAVELY, Google

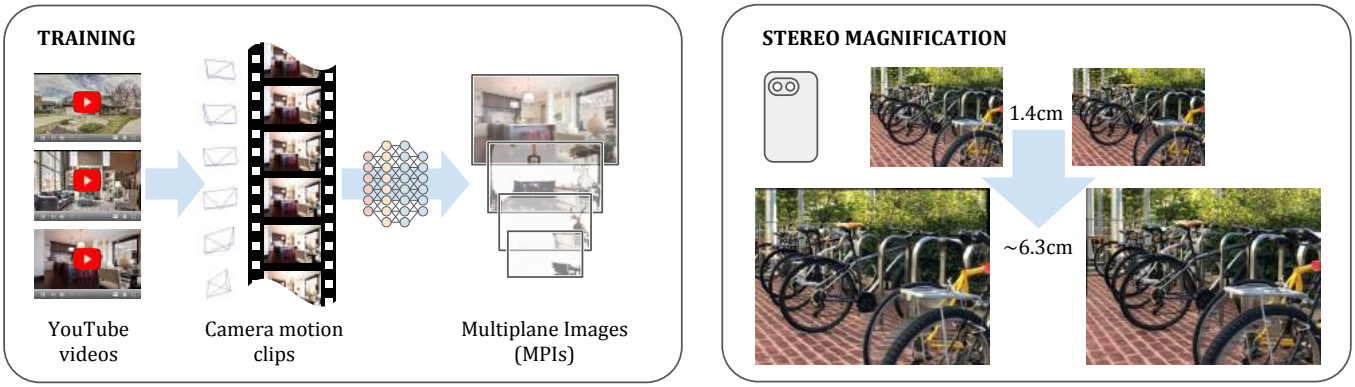



Fig. 1. We extract camera motion clips from YouTube videos and use them to train a neural network to generate a Multiplane Image (MPI) scene representation from narrow-baseline stereo image pairs. The inferred MPI representation can then be used to synthesize novel views of the scene, including ones that extrapolate significantly beyond the input baseline. (Video stills in this and other figures are used under Creative-Commons license from YouTube user *SonaVisual*.)

The view synthesis problem—generating novel views of a scene from known imagery—has garnered recent attention due in part to compelling applications in virtual and augmented reality. In this paper, we explore an intriguing scenario for view synthesis: extrapolating views from imagery captured by narrow-baseline stereo cameras, including VR cameras and now-widespread dual-lens camera phones. We call this problem *stereo magnification*, and propose a learning framework that leverages a new layered representation that we call *multiplane images* (MPIs). Our method also uses a massive new data source for learning view extrapolation: online videos on YouTube. Using data mined from such videos, we train a deep network that predicts an MPI from an input stereo image pair. This inferred MPI can then be used to synthesize a range of novel views of the scene, including views that extrapolate significantly beyond the input baseline. We show that our method compares favorably with several recent view synthesis methods, and demonstrate applications in magnifying narrow-baseline stereo images.

CCS Concepts: • **Computing methodologies** → **Computational photography**; **Image-based rendering**; *Neural networks*; *Virtual reality*;

Additional Key Words and Phrases: View extrapolation, deep learning

Authors' addresses: Tinghui Zhou, University of California, Berkeley; Richard Tucker, Google; John Flynn, Google; Graham Fyffe, Google; Noah Snavely, Google.


0730-0301/2018/8-ART65
https://doi.org/10.1145/3197517.3201323

**ACM Reference Format:**
Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. 2018. Stereo Magnification: Learning view synthesis using multiplane images . *ACM Trans. Graph.* 37, 4, Article 65 (August 2018), 12 pages. https://doi.org/10.1145/3197517.3201323

## 1 INTRODUCTION

Photography has undergone an upheaval over the past decade. Cellphone cameras have steadily displaced point-and-shoot cameras, and have become competitive with digital SLRs in certain scenarios. This change has been driven by the increasing image quality of cellphone cameras, through better hardware and also through computational photography functionality such as high dynamic range imaging [Hasinoff et al. 2016] and synthetic defocus [Apple 2016; Google 2017b]. Many of these recent innovations have sought to replicate capabilities of traditional cameras. However, cell phones are also rapidly acquiring new kinds of sensors, such as multiple lenses and depth sensors, enabling applications beyond traditional photography.

In particular, dual-lens cameras are becoming increasingly common. While stereo cameras have been around for nearly as long as photography itself, recently a number of dual-camera phones, such as the iPhone 7, have appeared on the market. These cameras tend to have a very small baseline (distance between views) on the order of a centimeter. We have also seen the recent appearance of a number of "virtual-reality ready" cameras that capture stereo images and

video from a pair of cameras spaced approximately eye-distance apart [Google 2017a].

Motivated by the proliferation of stereo cameras, our paper explores the problem of synthesizing new views from such narrow-baseline image pairs. While much prior work has explored the problem of *interpolating* between a set of given views [Chen and Williams 1993], we focus on the problem of *extrapolating* views significantly beyond the two input images. Such view extrapolation has many applications for photography. For instance, we might wish to take a narrow-baseline (~1cm) stereo pair on a cell phone and extrapolate to an IPD-separated (~6.3cm) stereo pair so as to create a photo with a compelling 3D stereo effect. Or, we might wish to take an IPD-separated stereo pair captured with a VR180 camera and extrapolate to an entire set of views along a line say half a meter in length, so as to enable full parallax with a small range of head motion. We call such view extrapolation from pairs of input views *stereo magnification*. The examples above involve magnifying the baseline by a significant amount—up to about 8x the original baseline.

The stereo magnification problem is challenging. We have just two views as input, unlike in common view interpolation scenarios that consider multiple views. We wish to be able to handle challenging scenes with reflection and transparency. Finally, we need the capacity to render pixels that are occluded and thus not visible in either input view. To address these challenges, our approach is to *learn* to perform view extrapolation from large amounts of visual data, following recent work on deep learning for view interpolation [Flynn et al. 2016; Kalantari et al. 2016]. However, our approach differs in key ways from prior work. First, we seek a scene representation that can be predicted once from a pair of input views, then reused to predict many output views, unlike in prior work where each output view must be predicted separately. Second, we need a representation that can effectively capture surfaces that are hidden in one or both input views. We propose a layered representation called a Multiplane Image (MPI) that has both of these properties. Finally, we need training data that matches our task. Simply collecting stereo pairs is not sufficient, because for training we also require additional views that are some distance from an input stereo pair as our ground truth. We propose a simple, surprising source for such data—online video, e.g., from YouTube, and show that large amounts of suitable data can be mined at scale for our task.

In experiments we compare our approach to recent view synthesis methods, and perform a number of ablation studies. We show that our method achieves better numerical performance on a held-out test set, and also produces more spatially stable output imagery since our inferred scene representation is shared for synthesizing all target views. We also show that our learned model generalizes to other datasets without re-training, and is effective at magnifying the narrow baseline of stereo imagery captured by cell phones and stereo cameras.

In short, our contributions include:

- A learning framework for stereo magnification (view extrapolation from narrow-baseline stereo imagery).
- Multiplane Images, a new scene representation for performing view synthesis.
- A new use of online video for learning view synthesis, and in particular view extrapolation.

## 2 RELATED WORK

*Classical approaches to view synthesis.* View synthesis—i.e., taking one or more views of a scene as input, and generating novel views—is a classic problem in computer graphics that forms the core of many image-based rendering systems. Many approaches focus on the interpolation setting, and operate by either interpolating rays from dense imagery ("light field rendering") [Gortler et al. 1996; Levoy and Hanrahan 1996], or reconstructing scene geometry from sparse views [Debevec et al. 1996; Hedman et al. 2017; Zitnick et al. 2004]. While these methods yield high-quality novel views, they do so by compositing the corresponding input pixels/rays, and typically only work well with multiple ($> 2$) input views. View synthesis from stereo imagery has also been considered, including converting 3D stereoscopic video to multi-view video suitable for glasses-free automultiscopic displays [Chapiro et al. 2014; Didyk et al. 2013; Kellnhofer et al. 2017; Riechert et al. 2012] and 4D light field synthesis from a micro-baseline stereo pair [Zhang et al. 2015], as well as generalizations that reconstruct geometry from multiple small-baseline views [Ha et al. 2016; Yu and Gallup 2014]. While we also focus on stereo imagery, the techniques we present can also be adapted to single-view and multi-view settings. We also target much larger extrapolations than prior work.

*Learning-based view synthesis.* More recently, researchers have applied powerful deep learning techniques to view synthesis. View synthesis can be naturally formulated as a learning problem by capturing images of a large number of scenes, withholding some views of each scene as ground truth, training a model that predicts such missing views from one or more given views, and comparing these predicted views to the ground truth as the loss or objective that the learning seeks to optimize. Recent work has explored a number of deep network architectures, scene representations, and application scenarios for learning view synthesis.

Flynn et al. [2016] proposed a view interpolation method called DeepStereo that predicts a volumetric representation from a set of input images, and trains a model using images of street scenes. Kalantari et al. [2016] use light field photos captured by a Lytro camera [Lytro 2018] as training data for predicting a color image for a target interpolated viewpoint. Both of these methods predict a representation in the coordinate system of the *target* view. Therefore, these methods must run the trained network for each desired target view, making real-time rendering a challenge. Our method predicts the scene representation once, and reuses it to render a range of output views in real time. Further, these prior methods focus on interpolation, rather than extrapolation as we do.

Other recent work has explored the problem of synthesizing a stereo pair [Xie et al. 2016], large camera motion [Zhou et al. 2016], or even a light field [Srinivasan et al. 2017] from a *single* image, an extreme form of extrapolation. Our work focuses on the increasingly common scenario of narrow-baseline stereo pairs. This two-view scenario potentially allows for generalization to more diverse scenes and larger extrapolation than the single-view scenario. The recent single-view method of Srinivasan et al., for instance, only considers

relatively homogeneous datasets such as macro shots of flowers, and extrapolates up to the small baseline of a Lytro camera, whereas our method is able to operate on diverse sets of indoor and outdoor scenes, and extrapolate views sufficient to allow slight head motions in a VR headset.

Finally, a variety of work in computer vision has used view synthesis as an indirect form of supervision for other tasks, such as predicting depth, shape, or optical flow from one or more images [Garg and Reid 2016; Godard et al. 2017; Liu et al. 2017; Tulsiani et al. 2017; Vijayanarasimhan et al. 2017; Zhou et al. 2017]. However, view synthesis is not the explicit goal of such work.

*Scene representations for view synthesis.* A wide variety of scene representations have been proposed for modeling scenes in view synthesis tasks. We are most interested in representations that can be predicted once and then reused to render multiple views at runtime. To achieve such a capability, representations are often volumetric or otherwise involve some form of *layering*. For instance, layered depth images (LDIs) are a generalization of depth maps that represent a scene using several layers of depth maps and associated color values [Shade et al. 1998]. Such layers allow a user to "see around" the foreground geometry to the occluded objects that lie behind. Zitnick et al., represent scenes using per-input-image depth maps, but also solve for alpha matted layers around depth discontinuities to achieve high-quality interpolation [2004]. Perhaps closest to our representation is that of Penner and Zhang [2017]. They achieve softness by explicitly modeling confidence, whereas we model transparency which leads to a different method of compositing and rendering. Additionally, whereas we build one representation of a scene, they produce a representation for each input view and then interpolate between them. Our representation is also related to the classic layered representation for encoding moving image sequences by Wang and Adelson [1994], and to the layered attenuators of Wetzstein, et al. [2011], who use actual physical printed transparencies to construct lightfield displays. Finally, Holroyd et al [2011] explore a similar representation to ours but in physical form.

The multiplane image (MPI) representation we use combines several attractive properties of prior methods, including handling of multiple layers and "softness" of layering for representing mixed pixels around boundaries or reflective/transparent objects. Crucially, we also found it to be suitable for learning via deep networks.

## 3 APPROACH

Given two images $I_1$ and $I_2$ with known camera parameters, our goal is to learn a deep neural net to infer a global scene representation suitable for synthesizing novel views of the same scene, and in particular extrapolating beyond the input views. In this section, we first describe our scene representation and its characteristics, and then present our pipeline and objective for learning to predict such representation. Note that while we focus on stereo input in this paper, our approach could be adapted to more general view synthesis setups with either single or multiple input views.

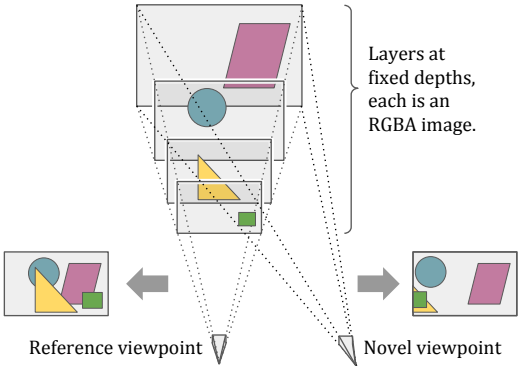



Fig. 2. An illustration of the multiplane image (MPI) representation. An MPI consists of a set of fronto-parallel planes at fixed depths from a reference camera coordinate frame, where each plane encodes an RGB image and an alpha map that capture the scene appearance at the corresponding depth. The MPI representation can be used for efficient and realistic rendering of novel views of the scene.

### 3.1 Multiplane image representation

The global scene representation we adopt is a set of fronto-parallel planes at a fixed range of depths with respect to a reference coordinate frame, where each plane $d$ encodes an RGB color image $C_d$ and an alpha/transparency map $\alpha_d$. Our representation, which we call a *Multiplane Image* (MPI), can thus be described as a collection of such RGBA layers $\{(C_1, \alpha_1), \ldots, (C_D, \alpha_D)\}$, where $D$ is the number of depth planes. An MPI is related to the *Layered Depth Image* (LDI) representation of Shade, et al. [Shade et al. 1998], but in our case the pixels in each layer are fixed at a certain depth, and we use an alpha channel per layer to encode visibility. To render from an MPI, the layers are composed from back-to-front order using the standard "over" alpha compositing operation. Figure 2 illustrates an MPI. The MPI representation is also related to the "selection-plus-color" layers used in DeepStereo [Flynn et al. 2016], as well as to the volumetric representation of Penner and Zhang [2017].

We chose MPIs because of their ability to represent geometry and texture including occluded elements, and because the use of alpha enables them to capture partially reflective or transparent objects as well as to deal with soft edges. Increasing the number of planes (which we can think of as increasing the resolution in disparity space) enables an MPI to represent a wider range of depths and allows a greater degree of camera movement. Furthermore, rendering views from an MPI is highly efficient, and could allow for real-time applications.

Our representation recalls the *multiplane camera* invented at Walt Disney Studios and used in traditional animation [Wikipedia 2017]. In both systems, a scene is composed of a series of partially transparent layers at different distances from the camera.

### 3.2 Learning from stereo pairs

We now describe our pipeline (see Figure 3) for learning a neural net that infers MPIs from stereo pairs. In addition to the input images $I_1$ and $I_2$, we take as input their corresponding camera parameters

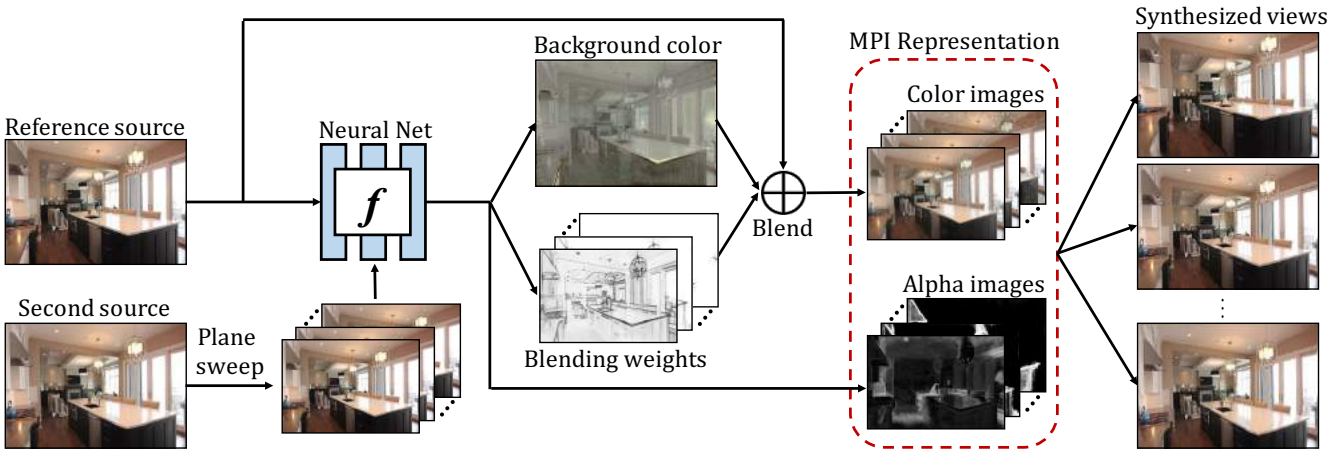



Fig. 3. Overview of our end-to-end learning pipeline. Given an input stereo image pair, we use a fully-convolutional deep network to infer the multiplane image representation. For each plane, the alpha image is directly predicted by the network, and the color image is blended by using the reference source and the predicted background image, where the blending weights are also output from the network. During training, the network is optimized to predict an MPI representation that reconstructs the target views using a differentiable rendering module (see Section 3.3). During testing, the MPI representation is only inferred once for each scene, which can then be used to synthesize novel views with minimal computation (homography + alpha compositing).

$c_1 = (p_1, k_1)$ and $c_2 = (p_2, k_2)$, where $p_i$ and $k_i$ denote camera extrinsics (position and orientation) and intrinsics, respectively.

The reference coordinate frame for our predicted scene is placed at the camera center of the first input image $I_1$ (i.e., $p_1$ is fixed to be the identity pose). Our training set consists of a large set of $\langle I_1, I_2, I_t, c_1, c_2, c_t \rangle$ tuples, where $I_t$ and $c_t = (p_t, k_t)$ denote the target ground-truth image and its camera parameters, respectively. We aim to learn a neural network, denoted by $f_\theta(\cdot)$, that infers an MPI representation using $\langle I_1, I_2, c_1, c_2 \rangle$ as input, such that when the MPI is rendered at $c_t$ it should reconstruct the target image $I_t$.

*Network input.* To encode the pose information from the second input image $I_2$, we compute a plane sweep volume (PSV) that reprojects $I_2$ into the reference camera at a set of $D$ fixed depth planes.[1] Although not required, we choose these depth planes to coincide with those of the output MPI. This plane sweep computation results in a stack of reprojected images $\{\hat{I}_2^1, \ldots, \hat{I}_2^D\}$, which we concatenate along the color channels, resulting in a $H \times W \times 3D$ tensor $\hat{I}_2$. We further concatenate $\hat{I}_2$ with $I_1$ to obtain the input tensor (of size $H \times W \times 3(D+1)$) to the network. Intuitively, the PSV representation allows the network to reason about the scene geometry by simply comparing $I_1$ to each planar reprojection of $I_2$—the scene depth at any given pixel is typically at the depth plane where $I_1$ and the reprojected $I_2$ agree. Many stereo algorithms work on this principle, but here we let the network automatically learn such relationships through the view synthesis objective.

*Network output.* A straightforward choice of the network output would be a separate RGBA image for each depth plane, where the color image captures the scene appearance and the alpha map encodes the visibility and transparency. However, such an output would be highly over-parameterized, and we found a more parsimonious output to be beneficial. In particular, we assume the color information in the scene can be well modeled by just two images, a foreground and a background image, where the foreground image is simply the reference source $I_1$, and the background image is predicted by the network, and is intended to capture the appearance of hidden surfaces. Hence, for each depth plane, we compute each RGB image $C_d$ as a per-pixel weighted average of the foreground image $I_1$ and the predicted background image $\hat{I}_b$:

$$C_d = w_d \odot I_1 + (1 - w_d) \odot \hat{I}_b \,, \tag{1}$$

where $\odot$ denotes the Hadamard product, and the blending weights $w_d$ are also predicted by the network. Intuitively, $I_1$ would have a higher weight at nearer planes where foreground content is dominant, while $\hat{I}_b$ is designed to capture surfaces that are occluded in the reference view. Note that the background image need not itself be a natural image, since the network can exploit the alpha and blending weights to selectively and softly use different parts of it at different depths. Indeed, there may be regions of a given background image that are never used in new views.

In summary, the network outputs the following quantities: 1) an alpha map $\alpha_d$ for each plane, 2) a global RGB background image $\hat{I}_b$ and 3) a blending weight image $w_d$ for each plane representing the relative proportion of the foreground and background layers at each pixel. If we predict $D$ depth layers each with a resolution of $W \times H$, then the total number of output parameters is $WH \cdot (2D + 3)$ (vs. $WH \cdot 4D$ for a direct prediction of an MPI). These quantities can then be converted to an MPI.

### 3.3 Differentiable view synthesis using MPIs

Given the MPI representation with respect to a reference frame, we can synthesize a novel view $\hat{I}_t$ by applying a planar transformation (inverse homography) to the RGBA image for each plane, followed

[1]For a rectified stereo pair, reprojected images would simply be shifted versions of $I_2$, though we consider more general configurations in our setup.

by a alpha-composition of the transformed images into a single image in a back-to-front order. Both the planar transformation and alpha composition are differentiable, and can be easily incorporated into the rest of the learning pipeline.

*Planar transformation.* Here we describe the planar transformation that inverse warps each MPI RGBA plane onto a target viewpoint. Let the geometry of the MPI plane to be transformed (i.e. the source) be $\mathbf{n} \cdot \mathbf{x} + a = 0$, where $\mathbf{n}$ denotes the plane normal, $\mathbf{x} = [u_s, v_s, 1]^T$ the source pixel homogeneous coordinates, and $a$ the plane offset. Since the source MPI plane is fronto-parallel to the reference source camera, we have $\mathbf{n} = [0, 0, 1]$ and $a = -d_s$, where $d_s$ is the depth of the source MPI plane. The rigid 3D transformation matrix mapping from source to target camera is defined by a 3D rotation $R$ and translation $\mathbf{t}$, and the source and target camera intrinsics are denoted $k_s$ and $k_t$, respectively. Then for each pixel $(u_t, v_t)$ in the target MPI plane, we use the standard inverse homography [Hartley and Zisserman 2003] to obtain

$$\begin{bmatrix} u_s \\ v_s \\ 1 \end{bmatrix} \sim k_s \left( R^T + \frac{R^T \mathbf{t} \mathbf{n} R^T}{a - \mathbf{n} R^T \mathbf{t}} \right) k_t^{-1} \begin{bmatrix} u_t \\ v_t \\ 1 \end{bmatrix} \tag{2}$$

Therefore, we can obtain the color and alpha values for each target pixel $[u_t, v_t]$ by looking up its correspondence $[u_s, v_s]$ in the source image. Since $[u_s, v_s]$ may not be an exact pixel coordinate, we use bilinear interpolation among the 4-grid neighbors to obtain the resampled values (following [Jaderberg et al. 2015; Zhou et al. 2016]).

*Alpha compositing.* After applying the planar transformation to each MPI plane, we then obtain the predicted target view by alpha compositing the color images in back-to-front order using the standard *over* operation [Porter and Duff 1984].

### 3.4 Objective

Given the MPI inference and rendering pipeline, we can train a network to predict MPIs satisfying our view synthesis objective. Formally, for a training set of $\langle I_1, I_2, I_t, c_1, c_2, c_t \rangle$ tuples, we optimize the network parameters by:

$$\min_{\theta} \sum_{\langle I_1, I_2, I_t, c_1, c_2, c_t \rangle} \mathcal{L}(\mathcal{R}(f_\theta(I_1, I_2, c_1, c_2), c_t), I_t) \,, \tag{3}$$

where $\mathcal{R}(\cdot)$ denotes the rendering pipeline described in Section 3.3 that synthesizes a novel view from the target camera $c_t$ using the inferred MPI $f_\theta(I_1, I_2, c_1, c_2)$, and $\mathcal{L}(\cdot)$ is the loss function between the synthesized view and the ground-truth. In this work, we use a deep feature matching loss (also referred to as the "perceptual loss" [Dosovitskiy and Brox 2016; Johnson et al. 2016; Zhang et al. 2018]), and specifically use the normalized VGG-19 [Simonyan and Zisserman 2014] layer matching from [Chen and Koltun 2017]:

$$\mathcal{L}(\hat{I}_t, I_t) = \sum_l \lambda_l \| \phi_l(\hat{I}_t) - \phi_l(I_t) \|_1 \,, \tag{4}$$

where $\{\phi_l\}$ is a set of layers in VGG-19 (`conv1_2`, `conv2_2`, `conv3_2`, `conv4_2`, and `conv5_2`) and the weight hyperparameters $\{\lambda_l\}$ are set to the inverse of the number of neurons in each layer.

Table 1. Our network architecture, where **k** is the kernel size, **s** the stride, **d** kernel dilation, **chns** the number of input and output channels for each layer, **in** and **out** are the accumulated stride for the input and output of each layer, and **input** denotes the input source of each layer with + meaning concatenation. See Section 3.5 for more details.

| Layer | k | s | d | chns | in | out | input |
|---|---|---|---|---|---|---|---|
| conv1_1 | 3 | 1 | 1 | 99/64 | 1 | 1 | $I_1 + \hat{I}_2$ |
| conv1_2 | 3 | 2 | 1 | 64/128 | 1 | 2 | conv1_1 |
| conv2_1 | 3 | 1 | 1 | 128/128 | 2 | 2 | conv1_2 |
| conv2_2 | 3 | 2 | 1 | 128/256 | 2 | 4 | conv2_1 |
| conv3_1 | 3 | 1 | 1 | 256/256 | 4 | 4 | conv2_2 |
| conv3_2 | 3 | 1 | 1 | 256/256 | 4 | 4 | conv3_1 |
| conv3_3 | 3 | 2 | 1 | 256/512 | 4 | 8 | conv3_2 |
| conv4_1 | 3 | 1 | 2 | 512/512 | 8 | 8 | conv3_3 |
| conv4_2 | 3 | 1 | 2 | 512/512 | 8 | 8 | conv4_1 |
| conv4_3 | 3 | 1 | 2 | 512/512 | 8 | 8 | conv4_2 |
| conv5_1 | 4 | .5 | 1 | 1024/256 | 8 | 4 | conv4_3 + conv3_3 |
| conv5_2 | 3 | 1 | 1 | 256/256 | 4 | 4 | conv5_1 |
| conv5_3 | 3 | 1 | 1 | 256/256 | 4 | 4 | conv5_2 |
| conv6_1 | 4 | .5 | 1 | 512/128 | 4 | 2 | conv5_3 + conv2_2 |
| conv6_2 | 3 | 1 | 1 | 128/128 | 2 | 2 | conv6_1 |
| conv7_1 | 4 | .5 | 1 | 256/64 | 2 | 1 | conv6_2 + conv1_2 |
| conv7_2 | 3 | 1 | 1 | 64/64 | 1 | 1 | conv7_1 |
| conv7_3 | 1 | 1 | 1 | 64/67 | 1 | 1 | conv7_2 |

### 3.5 Implementation details

Unless specified otherwise, we use $D = 32$ planes set at equidistant disparity (inverse depth) with the near and far planes at 1m and 100m, respectively.

*Network architecture.* We use a fully-convolutional encoder-decoder architecture (see Table 1 for detailed specification). The encoder pathway follows similar design as VGG-19 [Simonyan and Zisserman 2014], while the decoder consists of deconvolution (fractionally-strided convolution) layers with skip-connections from lower layers to capture fine texture details. Dilated convolutions [Chen et al. 2018; Yu and Koltun 2016] are also used in intermediate layers `conv4_1,2,3` to model larger scene context while maintaining the spatial resolution of the feature maps. Each layer is followed by a `ReLU` nonlinearity and layer normalization [Ba et al. 2016] except for the last layer, where `tanh` is used and no layer normalization is applied. Each of the last layer outputs (32 alpha images, 32 blending weight images, and 1 background RGB image) is further scaled to match the corresponding valid range (e.g. [0, 1] for alpha images).

*Training details.* We implement our system in TensorFlow [Abadi et al. 2016]. We train the network using the ADAM solver [Kingma and Ba 2014] for 600K iterations with learning rate $0.0002, \beta_1 = 0.9, \beta_2 = 0.999$, and batch size 1. During training, the images and MPI have a spatial resolution of $1024 \times 576$, but the model can be applied to arbitrary resolution at test time in a fully-convolutional manner. Training takes about one week on a Tesla P100 GPU.

## 4 DATA

For training we require triplets of images together with their relative camera poses and intrinsics. Creating such a dataset from scratch would require carefully capturing simultaneous photos of a variety of scenes from three or more appropriate viewpoints per scene. Instead, we identified an existing source of massive amounts of such data: video clips on YouTube shot from a moving camera. By sampling frames from such videos, we can obtain very large amounts of data comprising multiple views of the same scene shot from a variety of baselines. For this approach to work, we need to be able to identify suitable video clips, i.e., clips shot from a moving camera but with a static scene, with minimal artifacts such as motion blur or rolling-shutter distortion, and without other editing effects such as titles and overlays. Finally, given a suitable clip, we must estimate the camera parameters for each frame.

While many videos on YouTube are not useful for our purposes, we found a surprisingly large amount of suitable content, across several categories of video. One such category is real estate footage. Typical real estate videos feature a series of shots of indoor and outdoor scenes (the interior of a room or stairway, exterior views of a house, footage of the surrounding area, etc). Shots typically feature smooth camera movement and little or no scene movement. Hence, we decided to build a dataset from real estate videos as a large and diverse source of multi-view training imagery.

Accordingly, the rest of this section describes the dataset we collected, consisting of over 7,000 video clips from 1 to 10 seconds in length, together with the camera position, orientation and field of view for each frame in the sequence. To build this dataset, we devised a pipeline for mining suitable clips from YouTube. This pipeline consists of four main steps: 1) identifying a set of candidate videos to download, 2) running a camera tracker on each video to both estimate an initial camera pose for each frame and to subdivide the video into distinct shots/clips, 3) performing a full bundle adjustment to derive high-quality poses for each clip, and 4) filtering to remove any remaining unsuitable clips.

### 4.1 Identifying videos

We manually found a number of YouTube channels that published real estate videos exclusively or almost exclusively, and used the YouTube API to retrieve videos IDs listed under each channel. This yielded a set of approximately 1,500 candidate videos.

### 4.2 Identifying and tracking clips with SLAM

We wish to subdivide each video into individual clips, and identify clips that have significant camera motion. We found few readily available tools for performing camera tracking on arbitrary videos in the wild. Initially, we tried to use structure-from-motion methods developed in computer vision, such as Colmap [Schönberger and Frahm 2016]. These methods are optimized for photo collections, and we found them to be slow and prone to failure when applied to video sequences. Instead, we found that for our purposes we could adapt modern algorithms for SLAM (Simultaneous Localization and Mapping) developed in the robotics community.

Visual SLAM methods take as input a series of frames, and build and maintain a sparse or semi-dense 3D reconstruction of the scene while estimating the viewpoint of the current frame in a way consistent with this reconstruction. We use the ORB-SLAM2 system [Mur-Artal and Tardós 2015], though other methods could also apply [Engel et al. 2018; Forster et al. 2014].

SLAM algorithms are not designed to process videos containing multiple shots with cuts and dissolves between them, and they typically care only about the accuracy of the *current* frame's pose—in particular, as the scene is refined over time, earlier frames are not updated and may become inconsistent with the current state of the world. To deal with these issues, our approach is as follows: **1**. Feed successive frames of the video to ORB-SLAM2 as normal. **2**. When the algorithm reports that it has begun to track the camera, mark the start of a clip. **3**. When ORB-SLAM2 fails to track $K = 6$ consecutive frames, or when we reach a maximum sequence length $L$, consider the clip to have ended. **4**. Keeping the final scene model constant, reprocess all frames in the clip so as to estimate a consistent pose for each camera. **5**. Re-initialize ORB-SLAM2 so it is ready to start tracking a new clip on subsequent frames. In this way, we use ORB-SLAM2 not just to track frames, but also to divide a video into clips using tracking failure as a way to detect shot boundaries.

Since SLAM methods, including ORB-SLAM2, require known camera intrinsics such as field of view (which are unknown for arbitrary online videos), we simply assume a field of view of 90 degrees. This assumption worked surprisingly well for the purposes of identifying good clips. Finally, for the sake of speed, at this stage we process a lower resolution version of the video. The result of the above processing is a set of clips or sequences for each video, along with a preliminary set of camera parameters.

### 4.3 Refining poses with bundle adjustment

We next process each sequence at higher resolution, using a standard structure-from-motion pipeline to extract features from each frame, match these features across frames, and perform a global bundle adjustment using the Ceres non-linear least squares optimizer [Agarwal et al. 2016]. We initialize the cameras using the poses found by ORB-SLAM2, and add a weak penalty to the optimization that encourages the parameters not to stray too far from their initial values. The output for each sequence is a set of adjusted camera poses, an estimated field of view, and a sparse point cloud representing the scene. An example output is illustrated in Figure 4.

One difficulty with this process is that there is no way to determine global scene scale, so our reconstructed camera poses are up to an arbitrary scale per clip. This ambiguity will become important when we represent scenes with MPIs, because our representation is based on layers at specific depths, as described in Section 3.5. Hence, we "scale-normalize" each sequence using the estimated 3D point cloud, scaling it so that the nearest scene geometry is approximately a fixed distance from the cameras. In particular, for each frame we compute the 5th percentile depth among all point depths from that frame's camera. Computing this depth across all cameras in a sequence gives us a set of "near plane" depths. We scale the sequence so that the 10th percentile of this set of depths is 1.25m. (Recall that our MPI representation uses a near plane of 1m.)

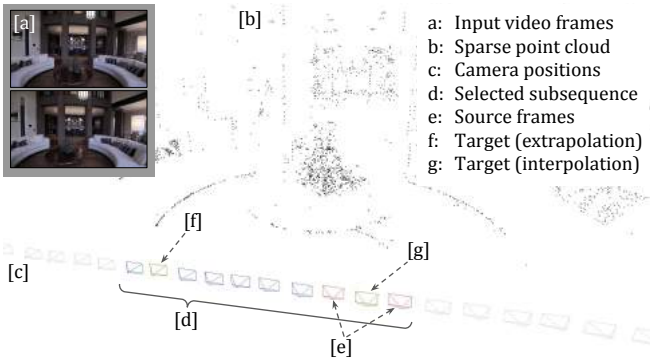



Fig. 4. Dataset output and frame selection, showing estimated camera trajectory and sparse point cloud. See section 4.5 for a detailed description.

### 4.4 Filtering and clipping

If the source video contains cross-fades, some frames may show a blend of two scenes. We discard ten frames from the beginning and end of each clip, which eliminates most such frames.

Occasionally the estimated camera poses for a sequence do not form a smooth track, which can indicate that we were unable to track the camera accurately. We define a frame to be *smooth* if its camera position $p_i$ is sufficiently close to the average of the two adjacent camera positions, specifically if $\|p_i - (p_{i+1} + p_{i-1})/2\| < 0.2 \times \|p_{i+1} - p_{i-1}\|$. For each sequence, we find the longest consecutive subsequence in which all frames are smooth, and discard the rest.

Finally we discard all remaining sequences of fewer than 30 frames. From an input set of approximately 1500 videos, this pipeline produces a set of ~7,000 sequences with a total of ~750K frames.

### 4.5 Choosing training triplets

Figure 4 shows an example of the result of this processing, including input video frames [a] (just two frames are shown here), and the sparse point cloud [b] and camera track [c] resulting from the structure from motion pipeline. As described in Section 3.2, for our application we require tuples $\langle I_1, I_2, I_t, c_1, c_2, c_t \rangle$, including cases where $I_t$ is an *extrapolation* from $I_1$ and $I_2$. We sample tuples from our dataset by first selecting from each sequence a random subsequence [d] of length 10, with stride (gap between selected frames) chosen randomly from 1 to 10. From this subsequence we then randomly choose two different frames and their poses to be the inputs $I_1, I_2, c_1$, and $c_2$ [e], and a third frame to be the target $I_t, c_t$.

Depending on which frames are chosen, the target frame may require extrapolation [f] (of up to a factor of nine times the distance between $I_1$ and $I_2$, assuming a linearly moving camera) or interpolation [g] from the inputs. We chose to learn to predict views from a variety of positions relative to the source imagery so as not to overfit to generating images at a particular distance during training.

## 5 EXPERIMENTS AND RESULTS

In this section we evaluate the performance of our method, and compare it with several view synthesis baselines. Our test set consists of 1,329 sequences that did not overlap with the training set.

Table 2. Quantitative comparison between our model and variants of the baseline Kalantari model [2016]. Higher SSIM/PSNR mean and lower rank are better. See Section 5.2 for more details.

| Method | Network | Loss | SSIM | | PSNR | |
|---|---|---|---|---|---|---|
| | | | Mean | Rank | Mean | Rank |
| Kalantari | Kalantari | pixel | 0.696 | 4.0 | 31.41 | 3.7 |
| Kalantari | Ours | VGG | 0.822 | 2.1 | 32.93 | 2.0 |
| Ours | Ours | Pixel | 0.812 | 2.6 | 32.42 | 2.8 |
| **Ours** | **Ours** | **VGG** | **0.835** | **1.4** | **33.10** | **1.5** |

For each sequence we randomly sample a triplet (two source frames and one target frame) for evaluation. We first visualize the MPI representation inferred by our model, and then provide detailed comparison with other recent view synthesis methods. We further validate our model design with various ablation studies, and finally highlight the utility of our method through several applications. For quantitative evaluation, we use the standard SSIM [Wang et al. 2004] and PSNR metrics.

### 5.1 Visualizing the multiplane images

We visualize examples of the MPI representation inferred by our network in Figure 5. Despite having no direct color or alpha ground-truth for each MPI plane during training, the inferred MPI is able to capture the scene appearance in a layer-wise manner (near to far) respecting the scene geometry, which allows realistic rendering of novel views from the representation.

We also demonstrate view extrapolation capability of the MPI representation in Figure 6, where we use the central two frames of a registered video sequence as input, and synthesize the previous and future frames with the inferred MPI. Please see the supplemental video for animations of these rendered sequences.

### 5.2 Comparison with Kalantari et al.

We compare our model with Kalantari et al. [2016], a state-of-the-art learning-based view synthesis method. A critical difference compared to our method is that Kalantari et al. has an *independent* rendering process for each novel view of the scene, and needs to re-run the entire inference pipeline every time a new view is queried, which is computationally prohibitive for real-time applications. In contrast, our method predicts a scene-level MPI representation that can render any novel viewpoint in real-time with minimal computation (inverse homography + alpha compositing).

We train and test two variants of their method on our data: 1) same network architecture (4 convolution layers) and pixel reconstruction loss from the original paper; 2) our network architecture (which is deeper with skip connections) with perceptual loss. For fair comparison, we use the same number of input planes as ours for constructing the plane sweep volume in their input. See Section 5.4 for discussion on the effect of varying the number of depth planes.

Table 2 shows mean SSIM and PSNR similarity metrics for each method across our test set. To measure if one method is consistently better than another, we also rank the methods on each test triplet and compute the average rank for each method. An average rank

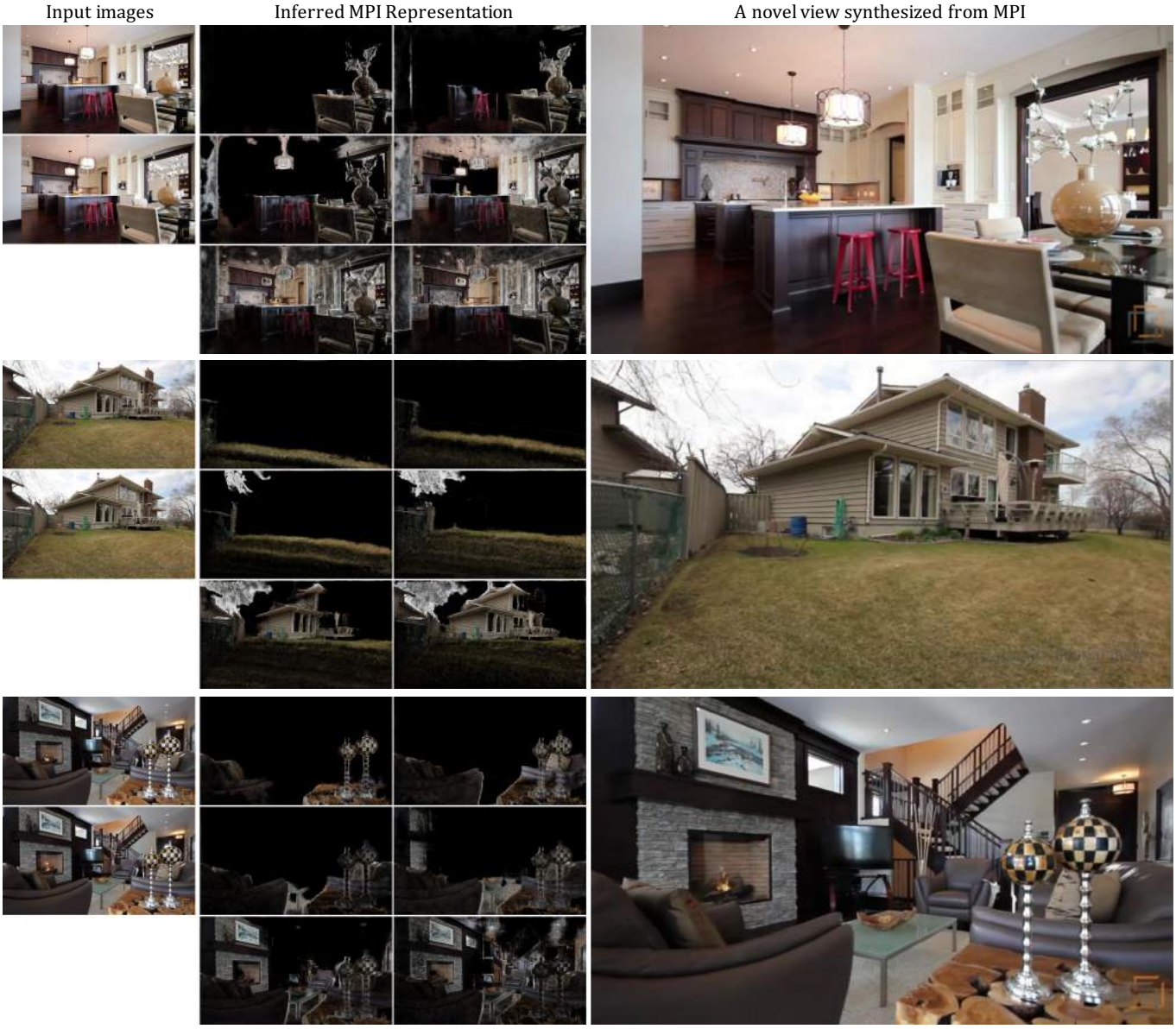



Fig. 5. Sample visualization of the input image pair (left), our inferred MPI representation (middle), where we show the alpha-multiplied color image at a subset of the depth planes from near to far (top to bottom, left to right), and novel views rendered from the MPI (right). The predicted MPI is able to capture the scene appearance in a layer-wise manner (near to far) respecting the scene geometry.

of 1.0 for PSNR, for example, would mean that this method always had the highest PSNR score.

We find that 1) our network architecture is significantly more effective than the simple 4-layer network used in the original Kalantari paper; 2) the VGG perceptual loss helps improve the performance over the pixel reconstruction loss (see Section 5.4 for discussion); 3) our model outperforms the better of the two Kalantari variants (VGG with our network architecture), indicating the high-quality of novel views rendered from the MPI representation.

We also observe that when rendering continuous view sequences of the same scene, our results tend to be more spatially coherent than Kalantari, and produce fewer frame-to-frame artifacts. We hypothesize that this is because, unlike the Kalantari model, we infer a single scene-level MPI representation that is shared for rendering all target views, which implicitly imposes a smoothness prior when rendering nearby views. Please see the video for qualitative comparisons of our method to Kalantari on rendered sequences.


Fig. 6. Sample view extrapolation results using multiplane images. The central two frames (green) are the input to our network, and the inferred MPI is used to render both past and future frames in the same video sequence.

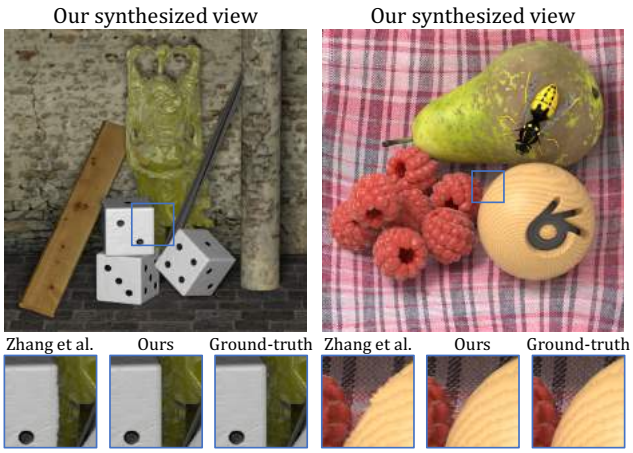



Fig. 7. Comparison with Zhang et al. [2015] on the HCI light field dataset [Wanner et al. 2013]. Note the differences around object boundaries.

## 5.3 Comparison with extrapolation methods

We compare with a non-learning view extrapolation approach by Zhang et al. [2015], which reconstructs a 4D light field from micro-baseline stereo pairs using disparity-assisted phase based synthesis (DAPS). For fair comparison, we directly apply our model trained on the real estate data to the HCI light field dataset [Wanner et al. 2013]. As shown in Figure 7, our model generalizes well on the HCI dataset without any fine-tuning, and compares favorably with Zhang et al. around depth boundaries, where our method introduces fewer distortion artifacts. We find that the method of Zhang et al. performs well for small view extrapolations, but breaks down more quickly around object boundaries with increasing extrapolation distance.

We also trained appearance flow [Zhou et al. 2016] on our dataset, but found rendered views exhibited significant artifacts, such as straight lines becoming distorted. This method appears more suited to object-centric synthesis than to scene rendering, and it is not able to fully exploit correlations between views since the trained network operates on each input image separately.

## 5.4 Ablation studies

*Perceptual loss.* To illustrate the effect of the perceptual loss, we compare our final model with a baseline model trained using L1 loss in the RGB pixel space. As shown in Figure 8, our final model trained using the perceptual loss better preserves object structure and texture details in the synthesized results than the baseline. The benefit of training with perceptual loss is further verified with quantitative evaluation in Table 2.

*Color layer prediction.* In Section 3.2, we propose that our network create the color values for each MPI plane as a weighted average of a network predicted "background" image and the reference source image. Here we compare several variants of the color prediction format (ordered by increasing level of representation flexibility):

(1) None. No color image or blending weights are predicted by the network. The reference source image is used as the color image at each MPI plane.
(2) Single image. The network predicts a single color image shared for all MPI planes.
(3) Background + blending weights (our preferred format). The network predicts a background image and blending weights. The reference source is used as the foreground image.
(4) Foreground + background + blending weights. In contrast to the previous variant, instead of using the reference source as the foreground image, the network predicts an extra foreground image for blending with the background.
(5) All images. The network directly outputs the color image at each MPI plane.

We compare the performance of these variants in Table 3 and show a qualitative example in Figure 9. Although "BG+blending weights" slightly outperforms the other variants, all the variants (other than "FG+BG+blending weights") produce competitive results. The "None" and "Single image" variants suffer in areas where the target view contains details that are occluded in the reference image but visible in the second input image. The "BG+blending weights" format can represent these areas better since not all MPI planes need to have the same color data. The "FG+BG+blending weights" variant is slightly more powerful as the foreground image is not restricted, and the "All images" variant, with a separate color image for each plane, is the only variant that can fully represent a scene with depth complexity

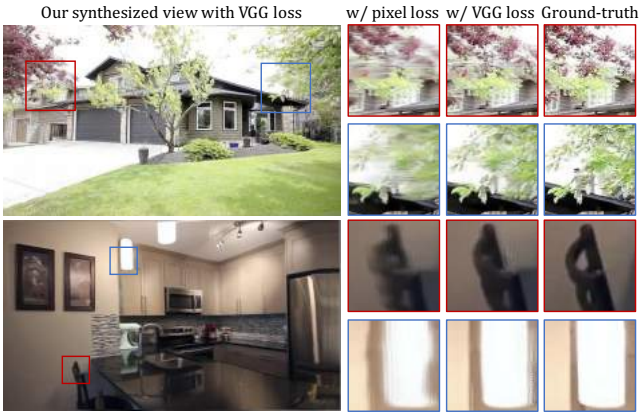



Fig. 8. Comparison between the models trained using pixel reconstruction loss and VGG perceptual loss. The latter better preserves object structure, and tends to produce sharper synthesized views.

Table 3. Quantitative evaluation of variants of network color output, ordered by increasing degree of flexibility (top to bottom). Higher SSIM/PSNR mean and lower rank are better.

| Color layer prediction | SSIM | | PSNR | |
|---|---|---|---|---|
| | Mean | Rank | Mean | Rank |
| None | 0.833 | 2.3 | 33.06 | 2.1 |
| Single image | 0.822 | 3.9 | 32.51 | 3.9 |
| **BG + Blend weights** | **0.835** | **1.6** | **33.09** | **1.6** |
| FG + BG + Blend weights | 0.819 | 4.1 | 32.50 | 3.7 |
| All images | 0.825 | 3.2 | 32.53 | 3.8 |

Table 4. Evaluating the effect of varying the number of depth planes for the MPI representation. Higher SSIM/PSNR mean and lower rank are better.

| MPI depth planes | SSIM | | PSNR | |
|---|---|---|---|---|
| | Mean | Rank | Mean | Rank |
| D = 8 | 0.766 | 2.99 | 32.12 | 2.96 |
| D = 16 | 0.812 | 1.98 | 32.73 | 1.97 |
| **D = 32** | **0.835** | **1.03** | **33.09** | **1.07** |

greater than 2. However, in our experiments these last two variants both performed slightly worse than "None". We hypothesize that the larger output space and less utilization of the reference image makes the learning harder with these output formats, and that the relatively small camera movement limits the depth complexity required.

*Number of depth planes.* As shown in Table 4, our model performance improves as more depth planes are used in the inferred MPI representation. We are currently limited to 32 planes due to memory constraints, but could overcome this with future hardware or alternative networks. As seen in Figure 10, the greater the offset between the reference view and the rendered view, the more planes are needed to render the scene accurately.

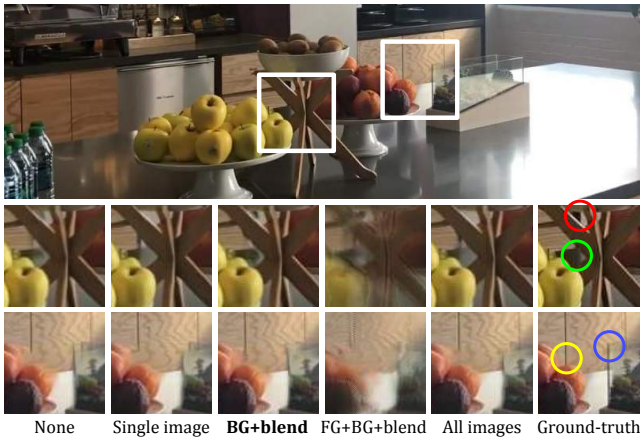



Fig. 9. Comparison between different color prediction formats. Note in particular the rendering of disoccluded background details, such as the rear wall (red), its reflection in the table surface (green), cupboard door (yellow) and corner of vase (blue). All the variants (except "FG+BG+blend") produce competitive results with slight differences. See Section 5.4 for more details.

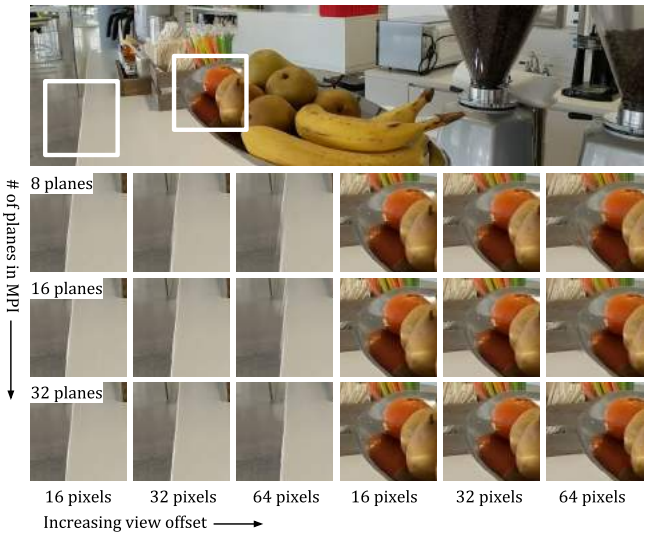
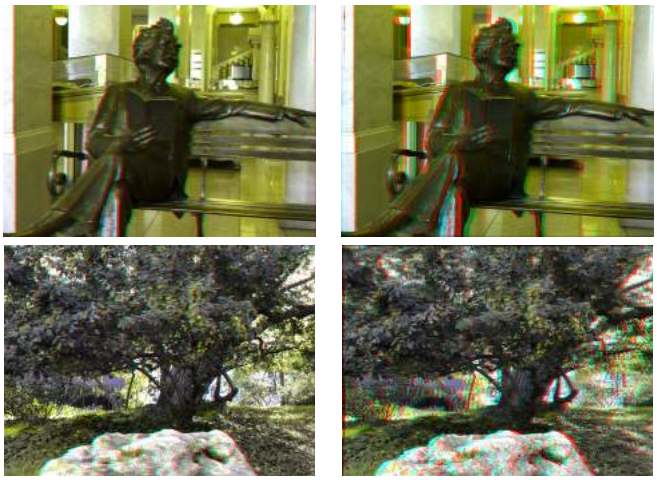



Fig. 10. Effect of varying the number of depth planes at different view offsets. For two regions of the top image, we show view extrapolations from MPIs with varying numbers of planes. The number of pixels shown is the disparity between front and back planes relative to the reference view. The larger the number of planes, the farther the view can be extrapolated before introducing artifacts. Note the edge of the counter in the first example, and the edges of objects in the second example. (Best viewed zoomed in.)

## 5.5 Applications

In this section we describe two applications of our trained model: 1) taking a narrow-baseline stereo pair from a cell phone camera and extrapolating to an average human interpupillary-distance (IPD)-spaced stereo pair, and 2) taking an image pair from a large-baseline stereo camera and extrapolating a "1D lightfield" of views between and beyond the source images.



Original baseline Magnified baseline

Fig. 11. Example stereo magnifications for dual-lens camera. Left: raw stereo pairs captured by an iPhone X, displayed as red-cyan anaglyph images, with a baseline of ~1.4cm. Right: the same images but with baseline synthetically magnified to ~6.3cm. Note the significantly enhanced stereo effect. (Best viewed zoomed in and with 3D glasses.)

*Cell phone image pairs → IPD stereo pair.* We captured a set of image pairs with an iPhone X, a recent dual-lens camera phone with a baseline of ~1.4cm, using an app that saves both captured views. Because the focal lengths of the two cameras are different, the app crops the wider-angle image to match the narrower field-of-view image. For each image pair, we ran a calibration procedure to refine the camera intrinsics using their nominal values as initialization. We then applied our model (trained on real estate data) to magnify the baseline to ~6.3cm (a magnification factor of 4.5x). Several results are shown in Figure 11 as anaglyph images, and in the supplemental video as sway animation. Figure 11 highlights how the extrapolated images provide a more compelling sense of 3D, and illustrates how our model can generalize to new scenarios that are atypical of real estate scenes (such as the sculpture of Mark Twain in the first example). Finally, notice that our method can handle interesting materials (e.g. the reflective glass and glossy floor in the first scene).

*Stereo pairs to extended 1D lightfield.* We also demonstrate taking a large-baseline stereo pair and synthesizing a continuous "1D lightfield"—i.e., a set of views along a line passing through the source views. For this application, we downloaded stereo pairs shot by a Fujifilm FinePix Real 3D W1 stereo point-and-shoot camera with a baseline of 7.7cm, and extrapolated to a continuous set of views with a baseline of 26.7cm (a magnification factor of ~3.5x). Figure 12 shows an example input and output as anaglyphs; see the supplemental video for animations of the resulting sequences. This input baseline, magnification factor, and scene content represent a challenging case for our model, and artifacts such as stretching in the background can be observed. Nonetheless, the results show plausible interpolations and extrapolations of the source imagery.

Original baseline Magnified baseline

Fig. 12. Example stereo magnifications for Fujifilm Real 3D stereo camera. Left: a raw stereo pair from the camera, displayed as red-cyan anaglyph images, with a baseline of ~7.7cm. Right: the same images but with baseline synthetically magnified to ~26.7cm. (Best viewed zoomed in and with 3D glasses.) (Photo used under CC license from Flickr user heiwa4126.)

## 6 DISCUSSION

Having trained on a large and varied dataset, our view synthesis system based on multiplane images is able to handle both indoor and outdoor scenes. We successfully applied it to scenes which are quite different from those in our training dataset. The learned MPIs are effective at representing surfaces which are partially reflective or transparent. Figure 13 (a) and (b) show two examples of such surfaces, rendered as anaglyphs with stereo-magnification.

Our method has certain limitations. When fine detail appears in front of a complex background, our model can struggle to place it at the correct depth. Figure 13 (c) shows a case where overhead cables appear to jump between two different depths. This may suggest that depth decisions are being made too locally. Figure 13 (d) shows the result of extrapolating beyond the limits of the MPI representation. When the disparity between adjacent layers exceeds one pixel we may see duplicated edges, producing a "stack of cards" effect.

In conclusion, we presented a new representation, training setup, and approach to learning view extrapolation from video data. We believe this framework can also generalize to a variety of different tasks, including extrapolating from more than two input images or from only one, and generating lightfields allowing view movement in multiple dimensions.

## REFERENCES


Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*.

Sameer Agarwal, Keir Mierle, and Others. 2016. Ceres Solver. http://ceres-solver.org. (2016).

Apple. 2016. Portrait mode now available on iPhone 7 Plus with iOS 10.1. https://www.apple.com/newsroom/2016/10/portrait-mode-now-available-on-iphone-7-plus-with-ios-101/. (2016).

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

Alexandre Chapiro, Simon Heinzle, Tunç Ozan Aydın, Steven Poulakos, Matthias Zwicker, Aljosa Smolic, and Markus Gross. 2014. Optimizing stereo-to-multiview conversion for autostereoscopic displays. In *Computer graphics forum*.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2018. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 40, 4 (2018).

Qifeng Chen and Vladlen Koltun. 2017. Photographic image synthesis with cascaded refinement networks. In *ICCV*.

Shenchang Eric Chen and Lance Williams. 1993. View Interpolation for Image Synthesis. In *Proc. SIGGRAPH*.

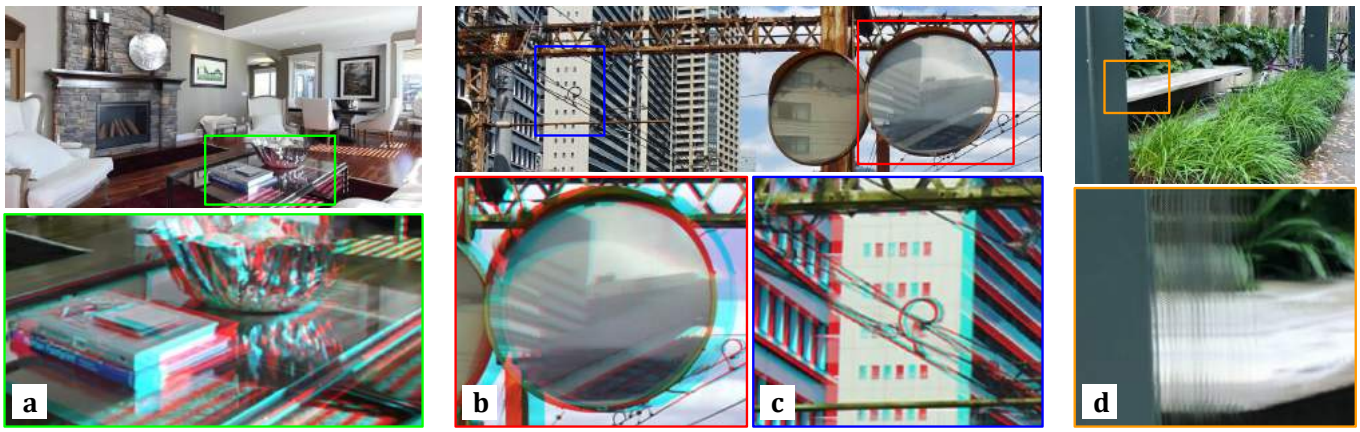



Fig. 13. Challenging cases. Reference images at top, rendered anaglyph details at bottom: (a) glass table with reflection and transparency, (b) reflection in a dusty curved mirror, (c) fine wires are confused with background, (d) extrapolation beyond the limits of the representation gives a 'stack of cards' effect.


Paul E Debevec, Camillo J Taylor, and Jitendra Malik. 1996. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proc. SIGGRAPH*.

Piotr Didyk, Pitchaya Sitthi-Amorn, William Freeman, Frédo Durand, and Wojciech Matusik. 2013. Joint view expansion and filtering for automultiscopic 3D displays. In *Proc. SIGGRAPH*.

Alexey Dosovitskiy and Thomas Brox. 2016. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*.

Jakob Engel, Vladlen Koltun, and Daniel Cremers. 2018. Direct sparse odometry. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 40, 3 (2018).

John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. 2016. DeepStereo: Learning to Predict New Views From the World's Imagery. In *CVPR*.

Christian Forster, Matia Pizzoli, and Davide Scaramuzza. 2014. SVO: Fast Semi-Direct Monocular Visual Odometry. In *ICRA*.

Ravi Garg and Ian Reid. 2016. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In *ECCV*.

Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. 2017. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In *CVPR*.

Google. 2017a. Introducing VR180 cameras. https://vr.google.com/vr180/. (2017).

Google. 2017b. Portrait mode on the Pixel 2 and Pixel 2 XL smartphones. https://research.googleblog.com/2017/10/portrait-mode-on-pixel-2-and-pixel-2-xl.html. (2017).

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The Lumigraph. In *Proc. SIGGRAPH*.

Hyowon Ha, Sunghoon Im, Jaesik Park, Hae-Gon Jeon, and In So Kweon. 2016. High-quality Depth from Uncalibrated Small Motion Clip. In *CVPR*.

Richard Hartley and Andrew Zisserman. 2003. *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. 2016. Burst photography for high dynamic range and low-light imaging on mobile cameras. In *Proc. SIGGRAPH Asia*.

Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. 2017. Casual 3D Photography. In *Proc. SIGGRAPH Asia*.

Michael Holroyd, Ilya Baran, Jason Lawrence, and Wojciech Matusik. 2011. Computing and fabricating multilayer models. In *Proc. SIGGRAPH Asia*.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. 2015. Spatial transformer networks. In *NIPS*.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*.

Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. 2016. Learning-Based View Synthesis for Light Field Cameras. In *Proc. SIGGRAPH Asia*.

Petr Kellnhofer, Piotr Didyk, Szu-Po Wang, Pitchaya Sitthi-Amorn, William Freeman, Fredo Durand, and Wojciech Matusik. 2017. 3DTV at Home: Eulerian-Lagrangian Stereo-to-Multiview Conversion. In *Proc. SIGGRAPH*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Marc Levoy and Pat Hanrahan. 1996. Light Field Rendering. In *Proc. SIGGRAPH*.

Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. 2017. Video Frame Synthesis Using Deep Voxel Flow. In *ICCV*.

Lytro. 2018. Lytro. https://www.lytro.com/. (2018).

Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. 2015. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Trans. on Robotics* 31, 5 (2015).

Eric Penner and Li Zhang. 2017. Soft 3D Reconstruction for View Synthesis. In *Proc. SIGGRAPH Asia*.

Thomas Porter and Tom Duff. 1984. Compositing Digital Images. In *Proc. SIGGRAPH*.

Christian Riechert, Frederik Zilly, Peter Kauff, Jens Güther, and Ralf Schäfer. 2012. Fully automatic stereo-to-multiview conversion in autostereoscopic displays. *The Best of IET and IBC* 4 (09 2012).

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *CVPR*.

Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. 1998. Layered depth images. In *Proc. SIGGRAPH*.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

Pratul P. Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. 2017. Learning to Synthesize a 4D RGBD Light Field from a Single Image. In *ICCV*.

Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. 2017. Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency. In *CVPR*.

Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. 2017. Sfm-net: Learning of structure and motion from video. *arXiv preprint arXiv:1704.07804* (2017).

John YA Wang and Edward H Adelson. 1994. Representing moving images with layers. *IEEE Trans. on Image Processing* 3, 5 (1994).

Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on Image Processing* 13, 4 (2004).

Sven Wanner, Stephan Meister, and Bastian Goldluecke. 2013. Datasets and benchmarks for densely sampled 4d light fields. In *VMV*.

G. Wetzstein, D. Lanman, W. Heidrich, and R. Raskar. 2011. Layered 3D: Tomographic Image Synthesis for Attenuation-based Light Field and High Dynamic Range Displays. In *Proc. SIGGRAPH*.

Wikipedia. 2017. Multiplane camera. https://en.wikipedia.org/wiki/Multiplane_camera. (2017).

Junyuan Xie, Ross B. Girshick, and Ali Farhadi. 2016. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In *ECCV*.

Fisher Yu and David Gallup. 2014. 3D Reconstruction from Accidental Motion. In *CVPR*.

Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *ICLR*.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Networks as a Perceptual Metric. In *CVPR*.

Zhoutong Zhang, Yebin Liu, and Qionghai Dai. 2015. Light field from micro-baseline image pair. In *CVPR*.

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

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

C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality Video View Interpolation Using a Layered Representation. In *Proc. SIGGRAPH*.