DeepView: View synthesis with learned gradient descent

John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe,
Ryan Overbeck, Noah Snavely, Richard Tucker
{jflynn,broxton,debevec,matthewduvall,fyffe,rover,snavely,richardt}@google.com
Google Inc.

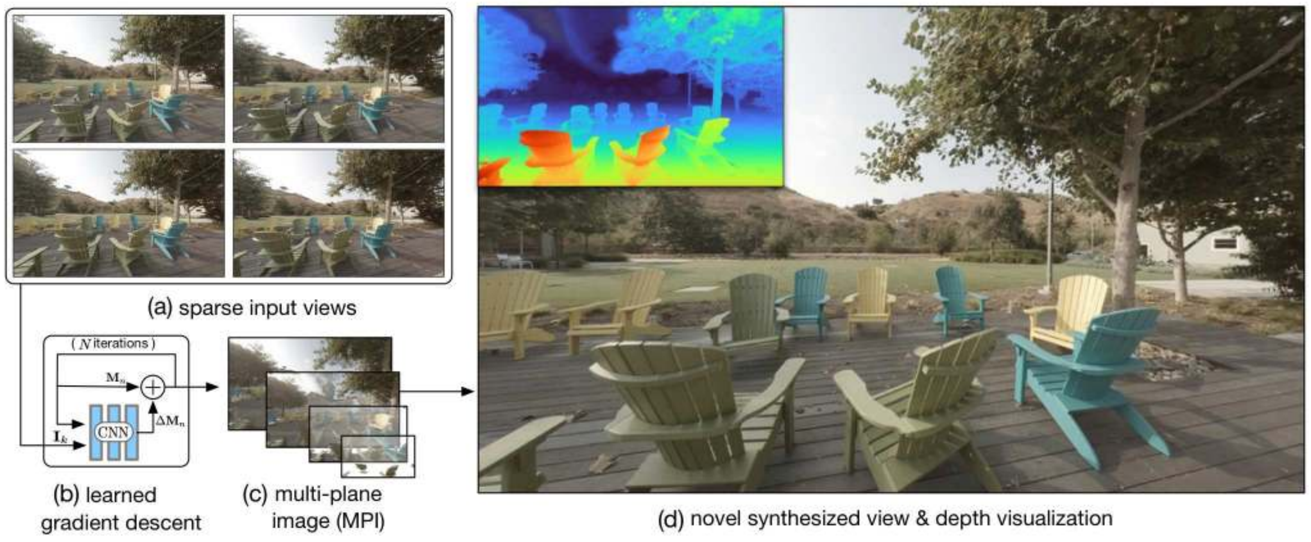

(a) sparse input views
(N iterations)
M_n
ΔM_n
I_k
CNN
(b) learned gradient descent
(c) multi-plane image (MPI)
(d) novel synthesized view & depth visualization

Figure 1: The DeepView architecture. **(a)** The network takes a sparse set of input images shot from different viewpoints. **(b, c)** The scene is reconstructed using learned gradient descent, producing a multi-plane image (a series of fronto-parallel, RGBA textured planes). **(d)** The multi-plane image is suitable for real-time, high-quality rendering of novel viewpoints. The result above uses four input views in a 30cm × 20cm rectangular layout. The novel view was rendered with a virtual camera positioned at the centroid of the four input views.

Abstract

We present a novel approach to view synthesis using multiplane images (MPIs). Building on recent advances in learned gradient descent, our algorithm generates an MPI from a set of sparse camera viewpoints. The resulting method incorporates occlusion reasoning, improving performance on challenging scene features such as object boundaries, lighting reflections, thin structures, and scenes with high depth complexity. We show that our method achieves high-quality, state-of-the-art results on two datasets: the Kalantari light field dataset, and a new camera array dataset, Spaces, *which we make publicly available.*

1. Introduction

Light fields offer a compelling way to view scenes from a continuous, dynamic set of viewpoints. The recently introduced *multiplane image* (MPI) representation approximates a light field as a stack of semi-transparent, colored layers arranged at various depths [32, 34] allowing real-time synthesis of new views of real scenes. MPIs are a powerful representation that can model complex appearance effects such as transparency and alpha matting.

Reconstructing an MPI from a sparse set of input views is an ill-posed *inverse problem*, like computed tomography (CT) or image deblurring, where we want to estimate a model whose number of parameters is much larger than the effective number of measurements. Such inverse problems can be solved using gradient descent based optimization methods that solve for the parameters that best predict the measurements via a forward model (such as a renderer in the case of view synthesis). However, for ill-posed problems these methods can overfit the measurements, necessitating the use of priors that are difficult to design and often data dependent. In the case of view synthesis, such overfitting results in synthesized views with noticeable visual artifacts.

Here we present DeepView, a new method for estimating a multiplane image from sparse views that uses *learned gradient descent* (LGD). Our method achieves high quality view synthesis results even on challenging scenes with thin objects or reflections. Building on recent ideas from the optimization community [2, 3, 5] this method replaces the simple gradient descent update rule with a deep network that generates parameter updates. The use of a learned update rule allows for a learned prior on the model parameters through manipulation of the gradients—effectively, the update network learns to generate representations that stay on the manifold of natural scenes. Additionally, the network learns to take larger, parameter-specific steps compared to standard gradient descent, leading to convergence in just a few iterations.

Our method, illustrated in Figure 1, takes as input a sparse set of views, e.g., from a camera rig (Figure 1(a)). It then processes the input images using learned gradient descent to produce an MPI (Figure 1(b,c)). Internally, this module uses a convolution neural network to predict an initial MPI from the input views, then iteratively improves this MPI via learned updates that take into account the current estimate. We show that for this problem the gradients have a particularly intuitive form in that they encode the visibility information between the input views and the MPI layers. By explicitly modeling visibility, our method shows improved performance in traditionally difficult areas such as edges and regions of high depth complexity. The resulting MPI can then be used for real-time view synthesis (Figure 1(d)).

In addition to the method itself, we also introduce a large, challenging dataset of light field captures called *Spaces*, suitable for training and testing view synthesis methods. On *Spaces* as well the standard light field dataset of Kalantari *et al*. [16], we show that our method produces high-quality view synthesis results that outperform recent approaches.

2. Background and related work

View synthesis and image-based rendering are classic problems in vision and graphics [8]. Given a set of input views, one way to pose view synthesis is to reconstruct a light field, a 4D function that directly represents all of the light rays that pass through a volume of space, from which one can generate any view within an appropriate region [19, 13]. However, it is rarely practical to measure a densely sampled light field. Instead, scenes are most often recorded using a limited number of viewpoints that sparsely sample the light field [20]. View synthesis research has focused on developing priors that aid in recovery of a dense light field from such sparse measurements.

Some prior view synthesis methods place explicit priors on the light field itself. For example, some degree of smooth view interpolation is made possible by assuming Lambertian reflectance [18], or that the light field is sparse in the Fourier [27] or Shearlet transform domain [30]. However, these approaches that do not explicitly model scene geometry can have difficulty correctly distinguishing between a scene's 3D structure and its reflectance properties. Other approaches explicitly reconstruct 3D geometry, such as a global 3D reconstruction [15] or a collection of per-input-view depth maps [37, 6, 20] using multi-view stereo algorithms. However, such methods based on explicit 3D reconstruction struggle with complex appearance effects such as specular highlights, transparency, and semi-transparent or thin objects that are difficult to reconstruct or represent.

In contrast, we use a scene representation called the multiplane image (MPI) [34] that provides a more flexible scene representation than depth maps or triangle meshes. Any method that predicts an MPI from a set of input images needs to consider the visibility between the predicted MPI and the input views so that regions of space that are occluded in some input views are correctly represented. In this regard, the task is similar to classic work on *voxel coloring* [26], which computes colored (opaque) voxels with an occlusion-aware algorithm. Similarly, Soft3D [21] improves an initial set of depth maps by reasoning about their relative occlusion.

One approach to generate an MPI would be to iteratively optimize its parameters with gradient descent, so that when the MPI is rendered it reproduces the input images. By iteratively improving an MPI, such an approach intrinsically models visibility between the MPI and input images. However, given a limited set of input views the MPI parameters are typically underdetermined so simple optimization will lead to overfitting. Hence, special priors or regularization strategies must be used [32], which are difficult to devise.

Alternatively, recent work has demonstrated the effectiveness of deep learning for view synthesis. Earlier methods, such as the so-called DeepStereo method [12] and the work of Kalantari *et al*. [16] require running a deep network for every desired output view, and so are less suitable for real-time view synthesis. In contrast, Zhou *et al*. predict an MPI from two input images directly via a learned feed-forward network [34]. These methods typically pass the input images to the network as a plane sweep volume (PSV) [11, 29] which removes the need to explicitly supply the camera pose, and also allows the network to more efficiently determine correspondences between images. However, such networks have no intrinsic ability to understand visibility between the input views and the predicted MPI, instead relying on the network layers to learn to model such geometric computations. Since distant parts of the scene may occlude each other the number of network connections required to effectively model this visibility can become prohibitively large.

In this work we adopt a hybrid approach combining the two directions—estimation and learning—described above: we model MPI generation as an inverse problem to be solved using a *learned* gradient descent algorithm [2, 3]. At inference time, this algorithm iteratively computes gradients of the current MPI with regard to the input images and processes the gradients with a CNN to generate an updated MPI. This update CNN learns to (1) avoid overfitting, (2) take large steps, thus requiring only a few iterations at inference time, and (3) reason about occlusions without dense connections by leveraging visibility computed in earlier iterations. This optimization can be viewed as a series of gradient descent steps that have been 'unrolled' to form a larger network where each step refines the MPI by incorporating the visibility computed in the previous iteration. From another perspective, our network is a recurrent network, that at each step is augmented with useful geometric operations (such as warping and compositing the MPI into different camera views) that help to propagate visibility information.

3. Method

We represent a three-dimensional scene using the recently introduced multiplane image representation [34]. An MPI, $\mathbf{M}$, consists of D planes, each with an associated $H \times W \times 4$ RGBA image. The planes are positioned at fixed depths with respect to a virtual reference camera, equally spaced according to inverse depth (disparity), in back-to-front order, $d_1, d_2, ..., d_D$, within the view frustum of the reference camera. The reference camera's pose is set to the centroid of the input camera poses, and its field of view is chosen such that it covers the region of space visible to the union of these cameras. We refer to the RGB color channels of the image for plane d as c_d and the corresponding alpha channel as α_d. To render an MPI to an RGB image $\tilde{\mathbf{I}}_k$ at a target view k we first warp the MPI images and then over-composite [22] the warped images from back to front:

$$\tilde{\mathbf{I}}_k = \mathcal{O}(\mathcal{W}_k(\mathbf{M})), \tag{1}$$

where the warp operator $\mathcal{W}_k(\mathbf{M})$ warps each of the D MPI images into view k's image space via a homography that is a function of the MPI reference camera, the depth of the plane, and the target view k [34]. The repeated over operator [22] $\mathcal{O}$ has a compact form, assuming premultiplied alpha c_d[1]:

$$\mathcal{O}(M) = \sum_{d=1}^{D} c_d \underbrace{\prod_{i=d+1}^{D} (1 - \alpha_i)}, \tag{2}$$

where M is an MPI warped to a view. We call the underbracketed term the *net transmittance* at the depth plane d, as it represents the fraction of the color that will remain after attenuation through the planes in front of d.

In this paper, we seek to solve the inverse problem associated with Eq. 1. That is, we wish to compute an MPI $\mathbf{M}$ that not only reproduces the input views, but also generates realistic novel views. Since the number of MPI planes is typically larger than the number of input images, the number of variables in an MPI will be much larger than the number of measurements in the input images.

3.1. Learned gradient descent for view synthesis

Inverse problems are often solved by minimization, e.g.:

$$\underset{\mathbf{M}}{\operatorname{argmin}} \sum_{k=1}^{K} \mathcal{L}_k(\mathbf{M}) + \Phi(\mathbf{M}), \tag{3}$$

[1]With premultiplied color the color channels are assumed to have been multiplied by α_{front}, so the two-image over operation reduces to $c_{\text{over}} = c_{\text{front}} + (1 - \alpha_{\text{front}})c_{\text{back}}$.

where $\mathcal{L}_k(\mathbf{M}) = \mathcal{L}(\mathbf{I}_k, \mathcal{O}(\mathcal{W}_k(\mathbf{M})))$ is a loss function measuring the disagreement between predicted and observed measurements, and Φ is a prior on $\mathbf{M}$. This non-linear optimization can be solved via iterative methods such as gradient descent, where the update rule (with step size λ) is given by:

$$\mathbf{M}_{n+1} = \mathbf{M}_n - \lambda \left[\sum_{k=1}^{K} \frac{\partial \mathcal{L}_k(\mathbf{M}_n)}{\partial \mathbf{M}_n} + \frac{\partial \Phi(\mathbf{M}_n)}{\partial \mathbf{M}_n} \right]. \tag{4}$$

Recent work on learned gradient descent (LGD) [2, 3] combines classical gradient descent and deep learning by replacing the standard gradient descent rule with a learned component:

$$\mathbf{M}_{n+1} = \mathbf{M}_n + \mathcal{N}_\omega \left(\frac{\partial \mathcal{L}_1(\mathbf{M}_n)}{\partial \mathbf{M}_n}, \ldots, \frac{\partial \mathcal{L}_K(\mathbf{M}_n)}{\partial \mathbf{M}_n}, \mathbf{M}_n \right), \tag{5}$$

where $\mathcal{N}_\omega$ is a deep network parameterized by a set of weights ω. (In practice, as described in Section 3.2, we do not need to explicitly specify $\mathcal{L}$ or compute full gradients.) The network processes the gradients to generate an update to the model's parameters. Notice that λ and Φ have been folded into $\mathcal{N}_\omega$. This enables $\mathcal{N}_\omega$ to learn a prior on $\mathbf{M}$ as well an adaptive, parameter-specific step size.

To train the network we unroll the N-iteration network obtaining the full network, denoted $\mathcal{M}_\omega$. We compute a loss by rendering the final MPI $\mathbf{M}_N$ to a held out view and comparing to the corresponding ground truth image $\mathbf{I}_{\text{gt}}$, under some training loss $\mathcal{L}_f$. The optimization of the network's parameters ω is performed over many training tuples $(\mathbf{I}_1, ..., \mathbf{I}_K, \mathbf{I}_{\text{gt}})$, across a large variety of scenes, using stochastic gradient descent:

$$\underset{\omega}{\operatorname{argmin}}\, \mathrm{E}\left[\mathcal{L}_f\left(\mathbf{I}_{\text{gt}}, \mathcal{O}(\mathcal{W}_{gt}(\mathcal{M}_\omega(\mathbf{I}_1, ..., \mathbf{I}_K)))\right)\right]. \tag{6}$$

After training, the resulting network can be applied to new, unseen scenes.

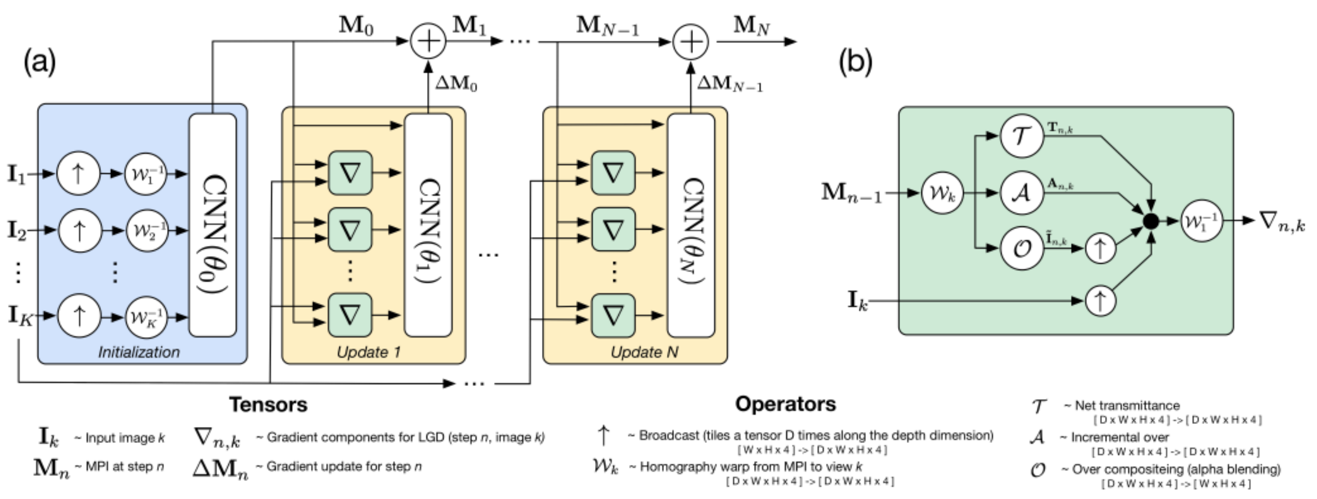
3.2. View synthesis gradients

LGD requires the gradients of the loss $\mathcal{L}(\tilde{\mathbf{I}}_k, \mathbf{I}_k)$ at each iteration. We show that for our problem, for any loss $\mathcal{L}$, the gradients have a simple interpretation as a function of a small set of components that implicitly encode scene visibility. We can pass these gradient components directly into the update networks, avoiding defining the loss explicitly.

The gradient of the k^{th} rendered image $\tilde{\mathbf{I}}_k = \mathcal{O}(\mathcal{W}_k(\mathbf{M}))$ is the combination, through the chain rule, of the gradient of the warping operation $\mathcal{W}_k$ and the gradient of the over operation $\mathcal{O}$. The warp operator's gradient is well approximated[2] by the inverse warp, $\mathcal{W}_k^{-1}$. The gradients of the over operation reduce to a particularly simple form:

$$\frac{\partial \mathcal{O}(M)}{\partial c_d} = \prod_{i=d+1}^{D} (1 - \alpha_i); \tag{7}$$

[2]They are exactly equivalent if the warp is perfectly anti-aliased.



(a)
$\mathbf{M}_0$
$\mathbf{M}_1$
$\mathbf{M}_{N-1}$
$\mathbf{M}_N$
$\Delta\mathbf{M}_0$
$\Delta\mathbf{M}_{N-1}$
$\mathbf{I}_1$
$\mathbf{I}_2$
$\mathbf{I}_K$
$\mathcal{W}_1^{-1}$
$\mathcal{W}_2^{-1}$
$\mathcal{W}_K^{-1}$
CNN(θ_0)
CNN(θ_1)
CNN(θ_N)
Initialization
Update 1
Update N
(b)
$\mathbf{M}_{n-1}$
$\mathcal{W}_k$
$\mathcal{T}$
$\mathbf{T}_{n,k}$
$\mathcal{A}$
$\mathbf{A}_{n,k}$
$\mathcal{O}$
$\tilde{\mathbf{I}}_{n,k}$
$\mathcal{W}_1^{-1}$
$\nabla_{n,k}$
$\mathbf{I}_k$
Tensors
$\mathbf{I}_k$ ~ Input image k
$\nabla_{n,k}$ ~ Gradient components for LGD (step n, image k)
$\mathbf{M}_n$ ~ MPI at step n
$\Delta\mathbf{M}_n$ ~ Gradient update for step n
Operators
$\uparrow$ ~ Broadcast (tiles a tensor D times along the depth dimension) [W x H x 4] -> [D x W x H x 4]
$\mathcal{W}_k$ ~ Homography warp from MPI to view k [D x W x H x 4] -> [D x W x H x 4]
$\mathcal{T}$ ~ Net transmittance [D x W x H x 4] -> [D x W x H x 4]
$\mathcal{A}$ ~ Incremental over [D x W x H x 4] -> [D x W x H x 4]
$\mathcal{O}$ ~ Over compositeing (alpha blending) [D x W x H x 4] -> [W x H x 4]

Figure 2: Our learned gradient descent network. **(a)** An initialization CNN generates an initial MPI based on the plane sweep volumes of the input images. A sequence of update CNNs generate updates to the MPI based on computed gradient components (detailed in **(b)**). All CNNs share the same core architecture (see Fig. 3) but are trained with different weights. **(b)** In place of the explicit loss gradient, we compute per-view gradient components, defined in Section 3.2. (The black circle represents channel-wise concatenation.)

$$\frac{\partial \mathcal{O}(M)}{\partial \alpha_d} = -\left[\sum_{i=1}^{d-1} c_i \prod_{j=i+1}^{d-1} (1-\alpha_j)\right]\left[\prod_{i=d+1}^{D} (1-\alpha_i)\right]. \tag{8}$$

We refer to the first bracketed term as the *accumulated over* at the depth slice d. It represents the repeated over of all depth slices behind the current depth slice. The second bracketed term is again the *net transmittance*. These per slice expressions can be stacked to form a 3D tensor containing the gradient w.r.t. all slices. We denote the operator that computes the accumulated over as $\mathcal{A}$ and the corresponding net transmittance operator as $\mathcal{T}$. Define the accumulated over computed in view k as $\mathbf{A}_k = \mathcal{A}(\mathcal{W}_k(\mathbf{M}))$ and the corresponding computed net transmittance as $\mathbf{T}_k = \mathcal{T}(\mathcal{W}_k(\mathbf{M}))$.

The gradient of *any* loss function w.r.t. to $\mathbf{M}$ will necessarily be some function of $\tilde{\mathbf{I}}_k$, $\mathbf{I}_k$, the accumulated over $\mathbf{A}_k$, and the net transmittance $\mathbf{T}_k$. Thus, without explicitly defining the loss function, we can write its gradient as some function of these inputs:

$$\frac{\partial \mathcal{L}_k(\mathbf{M})}{\partial \mathbf{M}} = \mathcal{F}(\mathcal{W}_k^{-1}([\mathbf{I}_k^{\uparrow}, \tilde{\mathbf{I}}_k^{\uparrow}, \mathbf{A}_k, \mathbf{T}_k])), \tag{9}$$

where the tensors between the square brackets are channel-wise concatenated and $^{\uparrow}$ represents the broadcast operator that repeats a 2D image to generate a 3D tensor. We define the *gradient components* $\nabla_{n,k} = \mathcal{W}_k^{-1}([\mathbf{I}_k^{\uparrow}, \tilde{\mathbf{I}}_k^{\uparrow}, \mathbf{A}_k, \mathbf{T}_k])$ and note that $\mathcal{W}_k^{-1}(\mathbf{I}_k^{\uparrow})$ and $\mathcal{W}_k^{-1}(\tilde{\mathbf{I}}_k^{\uparrow})$ are the familiar plane sweep volumes of the respective images.

In LGD the computed gradients are passed directly into the network $\mathcal{N}_\omega$; thus $\mathcal{F}$ is redundant—if needed it could be replicated by $\mathcal{N}_\omega$. Instead, we pass the gradient components $\nabla_{n,k}$ directly into $\mathcal{N}_\omega$:

$$\mathbf{M}_{n+1} = \mathbf{M}_n + \mathcal{N}_\omega\left(\nabla_{n,1}, \ldots, \nabla_{n,K}, \mathbf{M}_n\right). \tag{10}$$

Explicitly specifying the per-iteration loss is thus unnecessary. (However, we still must define a final training loss $\mathcal{L}_f$, as described in Section 3.3.) Instead, it is sufficient to provide the network with enough information such that it *could* compute the needed gradients. This flexibility, as shown by our ablation experiments in Section 4.2, allows the network to learn how to best use the rendered and input images during the LGD iterations. We note that this is related to the primal-dual method discussed in [2], where the dual operator is learned in the measurement space.

Our learned gradient descent network is visualized in Fig. 2, and may be intuitively explained as follows. We initialize the MPI by feeding the plane sweep volumes of the input images through an initialization CNN, similar to the early layers of other view synthesis networks such as Flynn *et al.* [12]. We then iteratively improve the MPI by computing and feeding the per-view gradient components into an update CNN. Interestingly, initializing using the plane sweep volume of the input images is equivalent to running one update step with the initial MPI slices set to zero color and $\alpha = 0$, further motivating the traditional use of plane sweep volumes in stereo and view synthesis.

Intuitively, since the gradient components contain the *net transmittance* and *accumulated over* they are useful inputs to the update CNN as they enable propagation of visibility information between the MPI and the input views. For example if a pixel in an MPI slice is occluded in a given view, as indicated by the value of the net transmittance, then the update network can learn to ignore this input view for that pixel. Similarly, the accumulated over for a view at a particular pixel within an MPI slice indicates how well the view is already explained by what is behind that slice. We note that by iteratively improving the visibility information our method shares similarities with the Soft3D method.

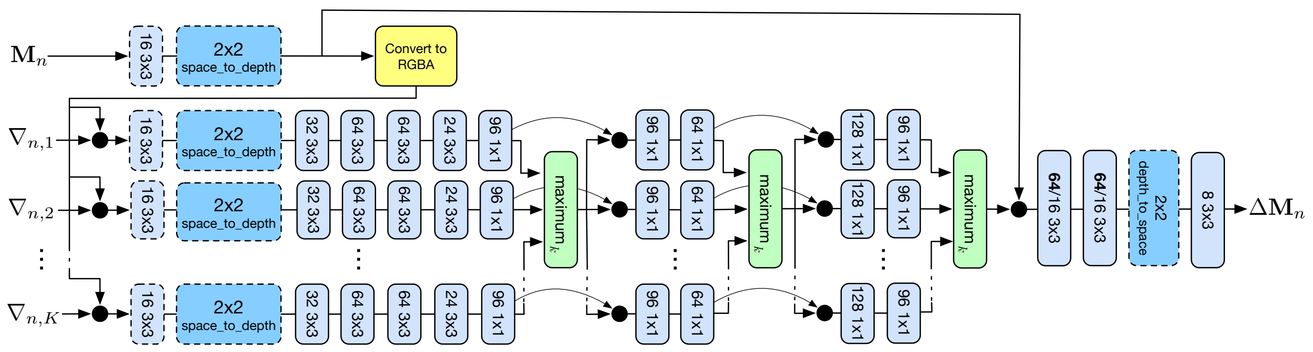


Figure 3: The DeepView update CNN. Convolutional layers are labeled with the number of filters followed by the kernel size. All convolutions use the Elu [10] activation, except the last which uses no activation. "**64**/16" indicates the use of 64 features for the initialization step and first iteration and 16 features on all subsequent iterations. Similarly, the downsampling operations with dotted outlines are only used during initialization and the first iteration. The maximum$_k$ operation computes the element-wise maximum across the k input tensors. Black circles represent channel-wise concatenation. The "Convert to RGBA" block applies a sigmoid activation to the first four channels of its input to generate valid RGBA values in the range $[0, 1]$, and then premultiplies the RGB channels by alpha. The initialization CNN has the same architecture, however there is no M_n input, and instead of the gradient components we input the PSVs of the input images.

3.3. Implementation

We now detail how we implement the described learned gradient descent scheme.

Per-iteration network architecture. For both the initialization and update networks we adopt a 2D convolutional neural network (CNN). The input to the CNN is the channel-wise concatenation of the current MPI and the computed gradient components (or, for the first iteration, the PSV of the input images). Within a single iteration the same 2D CNN (with the same parameters) runs on each depth slice, meaning the CNN is fully convolutional in all three of the MPI dimensions and allows changing both the resolution and the number of MPI depth planes after training. This allows performing inference at a high resolution and with a different number of depth planes.

We adapt recent ideas on aggregating across multiple inputs [23, 24, 35] to design our core per-iteration CNN architecture, shown in Fig. 3. We first concatenate each of the per-view gradient components $\nabla_{n,k}$ with the current MPI's RGBA values and transform it through several convolutional layers into a feature space. The per-view features are then processed through several stages that alternate between cross-view max-pooling and further convolutional layers to generate either an initial MPI or an MPI update. The only interaction between the views is thus through the max pool operation, leading to a network design that is independent of the order of the input views. Further, this network design can be trained with any number of input views in any layout. Intriguingly, this also opens up the possibility of a single network that operates on a variable number of input views with variable layout, although we have not yet explored this.

We use the same core CNN architecture for both the update and initialization CNNs, however, following [2] we use different parameters for each iteration. As in [2] we include extra channels (4 in our experiments) in addition to the RGBA channels.[3] These extra channels are passed from one iteration to the next, potentially allowing the model to emulate higher-order optimization methods such as LBFGS [36]. Additionally, we found that using a single level U-Net style network [25] (i.e. down-sampling the CNN input and up-sampling its output) for the first two iterations reduced RAM and execution time with a negligible effect on performance.

A major challenge for the implementation of our network was RAM usage. We reduce the RAM required by discarding activations within each of the per-iteration networks as described in [9], and tiling computation within the MPI volume. Additionally, during training, we only generate enough of the MPI to render a 32×32 pixel crop in the target image. More details are included in the supplemental material.

Training data. Each training tuple consists of a set of input views and a crop within a target view. We provide details of how we generate training tuples from the *Spaces* dataset in the supplemental material. For the dataset of Kalantari *et al*. [16], we follow the procedure described in their paper.

Our network design allows us to change the number of planes and their depths after training. However, the network may over-fit to the specific inter-plane spacing used during training. We mitigated this over-fitting by applying a random jitter to the depths of the MPI planes during training.

Training loss function. We use feature similarity [7, 33] as our training loss $\mathcal{L}_f$, specifically the *conv1_2*, *conv2_2* and *conv3_3* layers of a pre-trained VGG-16 network [28], and adopt the per layer scaling method discussed in [7].

Training parameters. We implemented our model in TensorFlow [1] and use the ADAM optimizer [17] with learning rate 0.00015. The supplemental material and Table 1 further describe the hyperparameters and the training setup used in our experiments.

[3]Note that in the discussion below "MPI" refers to the network's internal representation of the MPI, which includes these extra channels.

4. Evaluation

We evaluate our method on two datasets: the Lytro dataset of Kalantari *et al.* [16], and our own *Spaces* dataset. The Lytro dataset is commonly used in view synthesis research, but the baseline of each capture is limited by the small diameter of Lytro Illum's lens aperture. In contrast, view synthesis problems that stem from camera array captures are more challenging due to the larger camera separation and sparsely sampled viewpoints. We therefore introduce our new dataset, *Spaces*, to provide a more challenging shared dataset for future view synthesis research.

Spaces consists of 100 indoor and outdoor scenes, captured using a 16-camera rig (see Fig. 4). For each scene, we captured image sets at 5-10 slightly different rig positions (within ~10cm of each other). This jittering of the rig position provides a flexible dataset for view synthesis, as we can mix views from different rig positions for the same scene during training. We calibrated the intrinsics and the relative pose of the rig cameras using a standard structure from motion approach [14], using the nominal rig layout as a prior. We corrected exposure differences using a method similar to [4]. For our main experiments we undistort the images and downsampl them to a resolution of 800×480. For our ablation experiments, we used a lower resolution of 512×300 for expedience. We use 90 scenes from the dataset for training and hold out 10 for evaluation.

4.1. Quantitative Results

We compare our method to Soft3D and to a more direct deep learning approach based on Zhou *et al.* [34]. We also ran ablations to study the importance of different components of our model as well as the number of LGD iterations.

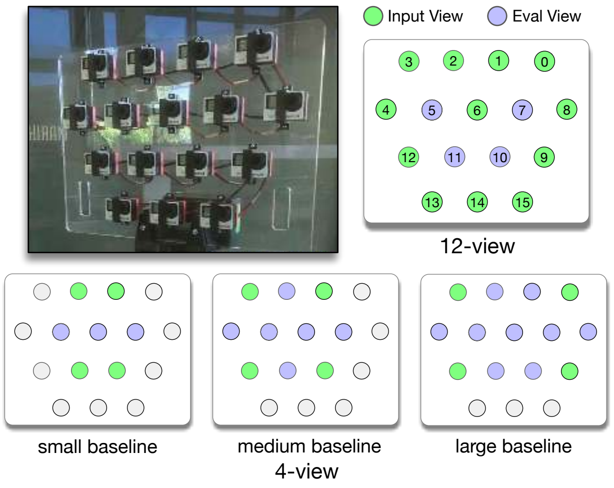


Figure 4: The *Spaces* dataset was captured using an array of 16 camera. The horizontal and diagonal distance between cameras is approximately 10cm. We experimented with different input and evaluation views, as shown. See Sec. 4 for more details.

Table 1: Experimental configurations. A check in the U-Net row indicates that the core CNN ran at a lower resolution for the first two LGD iterations. Other than "Iterations", all experiments used 3 LGD iterations. For some experiments we fine-tuned with a larger batch size for the last $20K$ iterations.

	Kalantari	4-View	12-View	Ablation	Iterations
Input Views	4	4	12	4	4
U-Net	✗	✓	✓	✓	✗
# Planes training	28	64	64	20	20
# Planes inference	28	80	80	20	20
Training iterations	300K	100K	100K	60K	80K
Batch size (fine tune)	32 (64)	20 (40)	20 (40)	32	32

Table 2: SSIM on the Kalantari Lytro data. (Higher is better.)

Scene	Soft3D	DeepView	Scene	Soft3D	DeepView
Flowers1	0.9581	**0.9668**	Rock	0.9595	**0.9669**
Flowers2	0.9616	**0.9700**	Leaves	0.9525	**0.9609**
Cars	0.9705	**0.9725**	(average)	0.9604	**0.9677**

We provide details on the different experimental setups in Table 1. In all experiments we measure image quality by comparing with the ground truth image using SSIM [31], which ranges between -1 and 1 (higher is better). For the ablation experiments we include the feature loss used during training, which is useful for relative comparisons (lower is better).

Results on the dataset of Kalantari *et al.* We train a DeepView model on the data from Kalantari *et al.* [16] using their described train-test split and evaluation procedure. As shown in Table 2, our method improves the average SSIM score by 18% (0.9674 vs. 0.9604) over the previous state-of-the-art, Soft3D [21].

Results on *Spaces*. The *Spaces* dataset can be evaluated using different input camera configurations. We trained three 4-view networks using input views with varying baselines, and a dense, 12-view network (as shown in Fig. 4). We found that our method's performance improves with the number of input views. When training all networks, target views were selected from all nearby jittered rig positions, although we evaluated only on views from the same rig as the input views, as shown in Fig. 5. For efficiency in some of our experiments we train on a lower number of planes than we used during inference, as shown in Fig. 5. The near plane depth for these experiments was set to 0.7m.

For comparison, the authors of Soft3D ran their algorithm on *S*paces. We show these results in Table 3. We also compare DeepView results to those produced with a variant of the network described in [34]. We adapt their network to use 4 wider baseline input views by concatenating the plane sweep volumes of the 3 secondary views onto the primary view (the top left of the 4 views), increasing the number of planes to 40, and increasing the number of internal features by 50%. Due to the RAM and speed constraints caused by the dense across-depth-plane connections it was not feasible

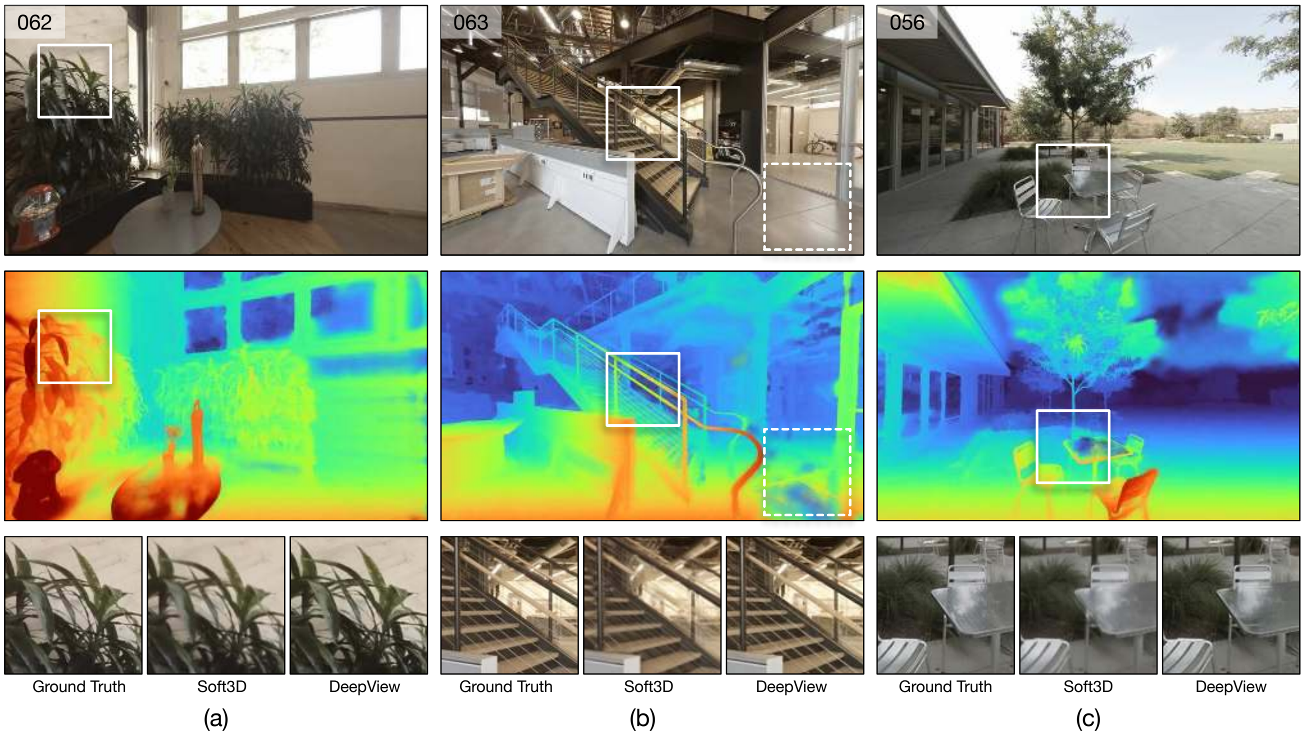


Figure 5: DeepView results on the Spaces dataset, 12-view configuration. **(Top row)** Synthesized views using 12 input views shown for evaluation camera viewpoint 7 (cf. Fig. 4). The top left index denotes the *Spaces* scene number. **(Middle row)** Depth visualizations produced from DeepView MPIs show the recovery of depth information in fine structures such as wires and foliage. **(Bottom row)** Each triplet shows a crop view comparing ground truth (left), Soft3D (center), and DeepView (right).

Table 3: SSIM on the *Spaces* dataset. [34]$^+$ represents the network adapted from [34].

Configuration	Soft3D[21]	[34]$^+$	DeepView 40 Plane	DeepView 80 Plane
4-view (small baseline)	0.9260	0.8884	0.9541	0.9561
4-view (medium baseline)	0.9300	0.8874	0.9544	0.9579
4-view (large baseline)	0.9315	0.8673	0.9485	0.9544
12-view	0.9402	n/a	0.9630	0.9651

to increase these parameters further, or to train a 12-view version of this model.

In all experiments our method yields significantly higher SSIM scores than both Soft3D and the Zhou *et al*.-based model [34]. On *Spaces*, we improve the average SSIM score of Soft3D by 39% (0.9584 vs. 0.9319). Additionally, our method maintains good performance at wider baselines and shows improvement as the number of input images increases from 4 to the 12 cameras used in the 12-view experiments.

The model based on [34] did not perform as well as DeepView, even when both algorithms were configured with the same number of MPI planes (see Table 3). This may because their original model was designed for a narrow-baseline stereo camera pair and relies on network connections to propagate visibility throughout the volume. As the baseline increases, an increasing number of connections are needed to propagate the visibility and performance suffers, as shown by the results for the largest baseline configuration.

4.2. Ablation and iteration studies

Gradient components. In this experiment we set one or more of the gradient components to zero to measure their importance to the algorithm. We also tested the effect of passing the gradient of an explicit L_2 loss during the LGD iterations, instead of the gradient components. Results are shown in Table 4. For experimental details see Table 1.

The best performance (in terms of feature loss) is achieved when all gradient components are included. When all components are removed the full network is equivalent to a residual network that operates on each depth slice independently, with no interaction across the depth slices. As expected, in this configuration the model performs poorly. Between these two extremes the performance decreases, as shown by both the SSIM and feature loss as well as the example images provided in the supplemental material. We note that our training optimizes the feature loss, and the drop in performance as measured by this loss, as opposed to SSIM, is significantly larger.

Number of iterations. We also measured the effect of varying the number of LGD iterations from 1 to 4. A single iteration corresponds to just the initialization network and the performance as expected is poor. The results improve as the number of iterations increases. We note that four iterations does show an improvement over three iterations. It

Table 4: Ablation study. Each run represents an experiment including the gradient components labeled as rendered image (`R`), transmittance (`T`), and accumulated over (`A`) with 3 iterations. The run labeled "∇L2" uses the true gradients of an L2 loss instead of gradient components. Experiments labeled "N" use `RTA` with the indicated number of iterations. Runs are sorted in order of descending SSIM; feature loss ($\mathcal{L}_f$) (lower is better) is also shown.

Run	SSIM	$\mathcal{L}_f$	Run	SSIM	$\mathcal{L}_f$	N	SSIM	$\mathcal{L}_f$
`R-A`	0.9461	1.196	`--A`	0.9397	1.271	4	0.9461	1.146
`RTA`	0.9446	1.179	∇L2	0.9389	1.250	3	0.9445	1.202
`RT-`	0.9434	1.232	`-T-`	0.9320	1.390	2	0.9417	1.242
`-TA`	0.9435	1.238	`---`	0.9075	1.765	1	0.8968	2.003
`R--`	0.9409	1.243						

would be interesting to test even more iterations. However, due to memory constraints, we were unable to test beyond four iterations, and even four iterations requires too much memory to be practical at higher resolutions.

4.3. Qualitative Results

We visually compare our method to both ground truth and Soft3D in Fig. 5 and in the supplemental material.We notice a general softness in Soft3D results as well as artifacts around edges. In contrast, DeepView produces convincing results in areas that are traditionally difficult for view synthesis including edges, reflections, and areas of high depth complexity. This is best seen in the interactive image comparison tool included in the supplemental material, which allows close examination of the differences between DeepView's and Soft3D's results.

In Fig. 5a our model produces plausible reflections on the table and convincing leaves on the plant where the depth complexity is high. In Fig. 5b our method reproduces the fine horizontal railings on the stairs which challenge previous work. Fig. 5c shows the crisp reconstruction of the complex foliage within the tree. Interestingly, DeepView can even render diffuse reflections, for example in Fig. 5b (dotted box). The way this is achieved can be seen in the corresponding depth map—transparent alpha values in the MPI permit the viewer to "see through" to a reflection that is rendered at a more distant plane.

The crop in Fig. 5c shows a difficult scene area for both our method and Soft3D. We note that *occluding* specular surfaces are particularly difficult to represent in an MPI. In this example, the MPI places the table surface on the far plane in order to mimic its reflective surface. However, the same surface should also occlude the chair legs which are closer. The end result is that the table surface becomes partially transparent, and the chair legs are visible through it.

Finally, we include a depth visualization in Fig. 5 produced by replacing the MPI's color channels with a false color while retaining the original α values.[4] This visualization shows the sharpness of the MPI around edges, even in complex areas such as tree branches.

[4]An interactive version of this visualization is included in the supplemental material.

5. Discussion

We have shown that a view synthesis model based on MPIs solved with learned gradient descent produces state-of-the-art results. Our approach can infer an MPI from a set of 4 input images in approximately 50 seconds on a P100 GPU. The method is flexible and allows for changing the resolution and the number and depth of the depth planes after training. This enables a model trained with medium distance scene objects to perform well even on scenes where more depth planes are needed to capture near objects.

Drawbacks and limitations: A drawback of our approach is the complexity of implementation and the RAM requirements and speed of training, which takes several days on multiple GPUs. The MPIs produced by our model share the drawback associated with all plane sweep volume based methods, in that the number of depth planes needs to increase with the maximum disparity. To model larger scenes it may be advantageous to use multiple MPIs and transition between them. Finally, our current implementation can only train models with a fixed number of input views, although our use of max-pooling to aggregate across views suggests the possibility of removing this restriction in the future.

Future work: Although our model is not trained to explicitly produce depth we were surprised by the quality of the depth visualizations that our model produces, especially around object edges. However, in smooth scene areas the visualization appears less accurate, as nothing in our training objective guides it towards the correct result in these areas. An interesting direction would be to include a ground truth depth loss during training.

The MPI is very effective at producing realistic synthesized images and has proven amenable to deep learning. However, it is over-parameterized for real scenes that consist of large areas of empty space. Enforcing sparsity on the MPI, or developing a more parsimonious representation with similar qualities, is another interesting area for future work.

6. Conclusion

We have presented a new method for inferring a multiplane image scene representation with learned gradient descent. We showed that the resulting algorithm has an intuitive interpretation: the gradient components encode visibility information that enables the network to reason about occlusion. The resulting method exhibits state-of-the-art performance on a difficult, real-world dataset. Our approach shows the promise of learned gradient descent for solving complex, non-linear inverse problems.

Acknowledgments We would like to thank Jay Busch and Matt Whalen for designing and building our camera rig, Eric Penner for help with Soft3D comparisons and Oscar Beijbom for some swell ideas.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5

[2] J. Adler and O. Öktem. Learned primal-dual reconstruction. *IEEE Transactions on Medical Imaging*, 37:1322–1332, 2018. 2, 3, 4, 5

[3] J. Adler and O. ktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 33(12):124007, 2017. 2, 3

[4] R. Anderson, D. Gallup, J. T. Barron, J. Kontkanen, N. Snavely, C. H. Esteban, S. Agarwal, and S. M. Seitz. Jump: Virtual reality video. 2016. 6

[5] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, 2016. 2

[6] G. Chaurasia, S. Duchêne, O. Sorkine-Hornung, and G. Drettakis. Depth synthesis and local warps for plausible image-based navigation. *Trans. on Graphics*, 32:30:1–30:12, 2013. 2

[7] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. *CoRR*, abs/1707.09405, 2017. 5

[8] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Proceedings of SIGGRAPH 93*, Annual Conference Series, 1993. 2

[9] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. 5

[10] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. 5

[11] R. T. Collins. A space-sweep approach to true multi-image matching. In *CVPR*, 1996. 2

[12] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to predict new views from the world's imagery. In *CVPR*, 2016. 2, 4

[13] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of SIGGRAPH 96*, Annual Conference Series, 1996. 2

[14] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003. 6

[15] P. Hedman, S. Alsisan, R. Szeliski, and J. Kopf. Casual 3d photography. *ACM Trans. Graph.*, 36(6):234:1–234:15, 2017. 2

[16] N. K. Kalantari, T.-C. Wang, and R. Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Trans. Graph.*, 35(6):193:1–193:10, 2016. 2, 5, 6

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 5

[18] A. Levin and F. Durand. Linear view synthesis using a dimensionality gap light field prior. In *CVPR*, 2010. 2

[19] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of SIGGRAPH 96*, Annual Conference Series, 1996. 2

[20] R. S. Overbeck, D. Erickson, D. Evangelakos, M. Pharr, and P. E. Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *CoRR*, abs/1810.08860, 2018. 2

[21] E. Penner and L. Zhang. Soft 3D reconstruction for view synthesis. *ACM Trans. Graph.*, 36(6):235:1–235:11, 2017. 2, 6, 7

[22] T. Porter and T. Duff. Compositing digital images. *SIGGRAPH Comput. Graph.*, 18(3):253–259, 1984. 3

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 5

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 5

[25] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). 5

[26] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *IJCV*, 35:151–173, 1997. 2

[27] L. Shi, H. Hassanieh, A. Davis, D. Katabi, and F. Durand. Light field reconstruction using sparsity in the continuous fourier domain. *Trans. on Graphics*, 34(1):12:1–12:13, Dec. 2014. 2

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 5

[29] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *IJCV*, 32(1), 1999. 2

[30] S. Vagharshakyan, R. Bregovic, and A. Gotchev. Light field reconstruction using shearlet transform. *IEEE Trans. PAMI*, 40(1):133–147, Jan. 2018. 2

[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc.*, 13(4):600–612, Apr. 2004. 6

[32] G. Wetzstein, D. Lanman, W. Heidrich, and R. Raskar. Layered 3D: Tomographic image synthesis for attenuation-based light field and high dynamic range displays. *ACM Trans. Graph.*, 30(4):95:1–95:12, 2011. 1, 2

[33] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018. 5

[34] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. Graph.*, 37(4):65:1–65:12, 2018. 1, 2, 3, 6, 7

[35] Y. Zhou and O. Tuzel. VoxelNet: End-to-end learning for point cloud based 3d object detection. *CVPR*, 2017. 5

[36] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-bfgs-b - fortran subroutines for large-scale bound constrained optimization. Technical report, ACM Trans. Math. Software, 1994. 5

[37] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004. 2