DeepStereo: Learning to Predict New Views from the World's Imagery

John Flynn
Zoox*
john.flynn@zoox.com

Ivan Neulander
Google Inc.
ineula@google.com

James Philbin
Zoox*
james@zoox.com

Noah Snavely
Google Inc.
snavely@google.com

Abstract

Deep networks have recently enjoyed enormous success when applied to recognition and classification problems in computer vision [22, 33], but their use in graphics problems has been limited ([23, 7] are notable recent exceptions). In this work, we present a novel deep architecture that performs new view synthesis directly from pixels, trained from a large number of posed image sets. In contrast to traditional approaches, which consist of multiple complex stages of processing, each of which requires careful tuning and can fail in unexpected ways, our system is trained end-to-end. The pixels from neighboring views of a scene are presented to the network, which then directly produces the pixels of the unseen view. The benefits of our approach include generality (we only require posed image sets and can easily apply our method to different domains), and high quality results on traditionally difficult scenes. We believe this is due to the end-to-end nature of our system, which is able to plausibly generate pixels according to color, depth, and texture priors learnt automatically from the training data. We show view interpolation results on imagery from the KITTI dataset [12], from data from [1] as well as on Google Street View images. To our knowledge, our work is the first to apply deep learning to the problem of new view synthesis from sets of real-world, natural imagery.

1. Introduction

Estimating 3D shape from multiple posed images is a fundamental task in computer vision and graphics, both as an aid to image understanding and as a way to generate 3D representations of scenes that can be rendered and edited. In this work, we aim to solve the related problem of *new view synthesis*, a form of image-based rendering (IBR) where the goal is to synthesize a new view of a scene by warping and combining images from nearby posed images. This can be used for applications such as cinematography, virtual reality, teleconferencing [4], image stabilization [21], or 3-dimensionalizing monocular film footage.

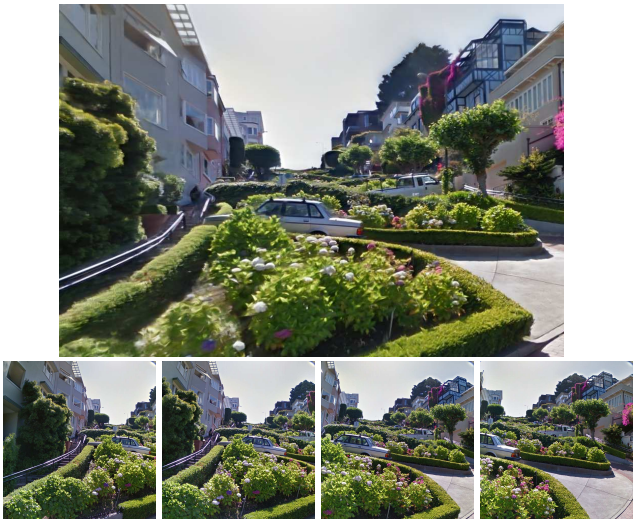

Figure 1: The top image was synthesized from several input panoramas. A portion of four of the inputs is shown on the bottom row.

New view synthesis is an extremely challenging, underconstrained problem. An exact solution would require full 3D knowledge of all visible geometry in the unseen view which is in general not available due to occluders. Additionally, visible surfaces may have ambiguous geometry due to a lack of texture. Therefore, good approaches to IBR typically require the use of strong priors to fill in pixels where the geometry is uncertain, or when the target color is unknown due to occlusions.

The majority of existing techniques for this problem involve traditional multi-view stereo and/or image warping methods and often explicitly model the stereo, color, and occlusion components of each target pixel [39, 1]. A key problem with these approaches is that they are prone to generating unrealistic and jarring rendering artifacts in the new view. Commonly seen artifacts include tearing around occluders, elimination of fine structures, and aliasing. Handling complex, self-occluding (but commonly seen) objects such as trees is particularly challenging for traditional ap-

*Contributed while at Google.

proaches. Interpolating between wide baseline views tends to exacerbate these problems.

Deep networks have enjoyed huge success in recent years, particularly for image understanding tasks [22, 33]. Despite these successes, relatively little work exists on applying deep learning to computer graphics problems and especially to generating new views from real imagery. One possible reason is the perceived inability of deep networks to generate pixels directly, but recent work on denoising [40], super-resolution [6], and rendering [23] suggests that this is a misconception. Another common objection is that deep networks have a huge number of parameters and are prone to overfitting in the absence of enormous quantities of data, but recent work [33] has shown state-of-the-art deep networks whose parameters number in the low millions greatly reducing the potential for overfitting.

In this work we present a new approach to new view synthesis that uses deep networks to regress directly to output pixel colors given posed input images. Our system is more resilient to the failure modes of traditional approaches, and is able to interpolate between views separated by a wide baseline. We posit this is due to the end-to-end nature of the training, and the ability of deep networks to learn extremely complex non-linear functions of their inputs [28]. Our method makes minimal assumptions about the scene being rendered: largely, that the scene should be static and should exist within a finite range of depths. Even when these requirements are violated, the resulting images degrade gracefully and often remain visually plausible. When uncertainty cannot be avoided, our method prefers to blur detail, which generates much more visually pleasing results compared to tearing or repeating, especially when animated. Additionally, although we focus on its application to new view problems here, we believe that the deep architecture presented can be readily applied to other stereo and graphics problems given suitable training data.

For the view synthesis problem, there is an abundance of readily available training data—any set of posed images can be used as a training set by leaving one image out and trying to reproduce it from the remaining images. Based on this key idea, we train two models using two corpuses: large amounts of data mined from Google's Street View, a massive collection of posed imagery spanning much of the globe [18], and posed imagery from the KITTI odometry dataset [12], which we repurpose for view synthesis. Because of the variety of Street View scenes seen during training, our system is robust and generalizes to new types of imagery, as well as to image collections used in prior work. Suprisingly, even a model trained on the less varied KITTI datset generalizes well to very different data.

To evaluate our method, we use held-out KITTI imagery to form a new view synthesis benchmark, and perform quantitative comparisons to more traditional IBR methods using this benchmark. We also qualitatively compare images generated by our model with the corresponding captured images, and compare our results qualitatively to existing state-of-the-art IBR methods [1].

2. Related Work

Learning depth from images. The problem of view synthesis is strongly related to the problem of predicting depth or 3D shape from imagery. In recent years, learning methods have been applied to this shape prediction problem, often from just a single image—a very challenging vision task. Automatic single-view methods include the Make3D system of Saxena *et al*. [30], which uses aligned photos and laser scans as training data, and the automatic photo pop-up work of Hoiem *et al*. [15], which uses images with manually annotated geometric classes. More recent methods have used Kinect data for training [17, 20] and deep learning methods for single view depth or surface normal prediction [9, 38]. However, the single-view problem remains very challenging. Moreover, gathering sufficient training data is difficult and time-consuming.

Other work has explored the use of machine learning for the stereo problem (i.e., using more than one frame). Learning has been used to estimate the parameters of more traditional models such as MRFs [43, 41], as well as for deriving low-level correlation filters for disparity estimation [27, 19]. Zbontar and LeCun [42] train a convolutional network on KITTI data to predict image patch similarity. This patch similarity network is used as the basis of a stereo matching cost, which, combined with traditional stereo filtering, achieves impressive results. Unlike this prior work, we learn to synthesize new views directly using a new deep architecture, and do not require known depth or disparity as training data.

View interpolation. There is a long history of work on image-based rendering in vision and graphics based on a variety of methods, including light fields [25, 14], image correspondence and warping [31], and explicit shape and appearance estimation [37, 44, 32]. Much of the recent work in this area has used a combination of 3D shape with image warping and blending [10, 13, 1, 2]. These methods are largely hand-built and do not leverage training data. Our goal is to learn a model for predicting new viewpoints by directly minimizing the prediction error on our training set.

We are particularly inspired by the work of Fitzgibbon *et al*. on IBR using image-based priors [11]. Like them, we consider the goal of faithfully reconstructing the output image to be the key problem to be optimized for, as opposed to reconstructing depth or other intermediate representation. We use state-of-the-art machine learning methods with a new architecture to achieve this goal. Szeliski [34] suggests image prediction as a metric for stereo algo-

rithms; our method directly minimizes this prediction error.

Finally, a few recent papers have applied deep learning to synthesizing imagery. Dosovitskiy *et al*. train a network on synthetic images of rendered 3D chairs that can generate new chair images given parameters such as pose [7]. Kulkarni *et al*. propose a "deep convolutional inverse graphics network" that can parse and rerender imagery such as faces [24]. However, we believe ours is the first method to apply deep learning to synthesizing novel natural imagery from posed real-world input images.

3. Approach

Given a set of N posed input images $I_1, I_2, \ldots, I_N$, with poses $V_1, V_2, \ldots, V_N$, the view synthesis problem is to render a new image from the viewpoint of a new target camera C. Despite the representative power of deep networks, naively training a deep network to synthesize new views by supplying the input images I_k as inputs directly is unlikely to work well, for two key reasons.

First, the pose parameters of C and of the views $V_1, V_2, \ldots, V_N$ would need to be supplied as inputs to the network in order to produce the desired view. The relationship between the pose parameters, the input pixels and the output pixels is complex and non-linear—the network would effectively need to learn how to interpret rotation angles and perform image reprojection. Forcing the network to learn projection is inefficient—it is a straightforward operation that we can represent outside of the network.

Second, in order to synthesize a new view, the network would need to compare and combine potentially distant pixels in the original source images, necessitating very dense, long-range connections. Such a network would have many parameters and would be slow to train, prone to overfitting, and slow to run inference on. A network structure could be designed to use the epipolar constraint internally in order to limit connections to those on corresponding epipolar lines. However, the epipolar lines, and thus the network connections, would be pose-dependent, making this very difficult and likely computationally inefficient in practice.

Using plane-sweep volumes. Instead, we address these problems by using ideas from traditional plane sweep stereo [3, 35]. We provide our network with a set of 3D plane sweep volumes as input. A plane sweep volume (PSV) consists of a stack of images reprojected to the target camera C (Fig. 2). Each image I_k in the stack is reprojected into C at a set of varying depths $d \in \{d_1, d_2, \ldots d_D\}$ to form a plane sweep volume $V_C^k = \{P_1^k, P_2^k, \ldots P_D^k\}$, where P_i^k refers to the reprojected image I_k at depth d_i. Reprojecting an input image into a target camera requires basic texture mapping and can be performed on a GPU. We create a separate plane sweep volume V_C^k for each input image I_k. Each voxel $v_{i,j,z}^k$ in each plane sweep volume V_C^k has RGB and A (alpha) components. The alpha channel indicates the availability of source pixels for that voxel (e.g., alpha $= 0$ for pixels outside the field of view of a source image).

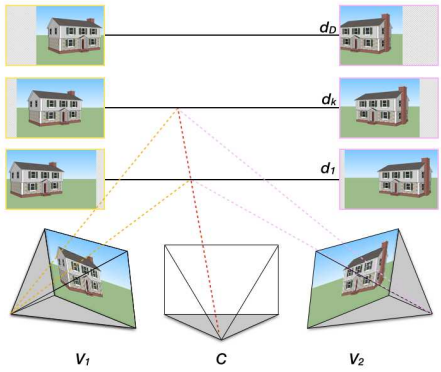

Figure 2: Plane sweep stereo reprojects images I_1 and I_2 from viewpoints V_1 and V_2 to the target camera C at a range of depths $d \in d_1 \ldots d_D$. The dotted rays indicate the pixels from the input images reprojected to a particular output image pixel, and the images above each input view show the corresponding reprojected images at different depths.

Using plane sweep volumes as inputs to the network removes the need to supply the pose parameters since they are now implicit inputs used in the construction of the PSV. Additionally, the epipolar constraint is trivially enforced within a PSV: corresponding pixels are now in corresponding i, j columns of the PSV. Thus, long-range connections between pixels are no longer needed, so a given output pixel depends only on a small column of voxels from each of the per-source PSVs. Similarly, the computation performed to produce an output pixel p at location i, j should be largely independent of the pixel location. This allows us to use more efficient convolutional neural networks. Our model applies 2D convolutional layers to each plane within the input PSV. In addition to sharing weights within convolutional layers, we make extensive use of weight sharing across planes in the PSV. Intuitively, weight sharing across planes makes sense since the computation to be performed on each plane will be largely independent of the plane's depth.

Our model. Our network architecture (Fig. 3) consists of two towers of layers, a *selection tower* and a *color tower*. The intuition behind this dual architecture is that there are really two related tasks we seek to accomplish:

- **Depth prediction.** First, we want to know the approximate depth for each pixel in the output image. This enables us to determine the source image pixels we should use to generate that output pixel. In prior work, this kind of probability over depth might be computed via SSD, NCC, or variance; we learn how to compute these probabilities using training data.

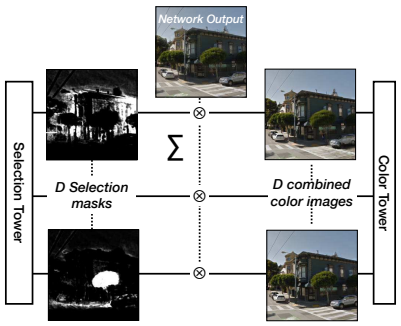

Figure 3: The basic architecture of our network, with selection and color towers. The final output image is produced by element-wise multiplication of the selection and color tower outputs and then computing the sum over the depth planes. Fig. 6 shows the full network details.

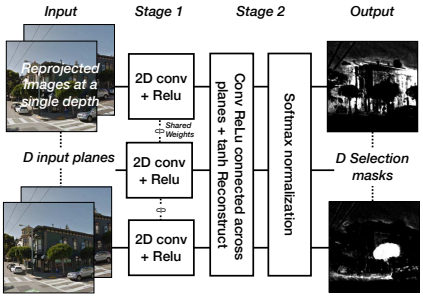

Figure 4: The selection tower learns to produce a selection probability $s_{i,j,z}$ for each pixel $p_{i,j}$ in each depth plane P_z. The first 2D layer operates on the individual reprojected images. Subsequent layers operate on the concatenated features per depth plane.

- **Color prediction.** Second, we want to produce a color for that output pixel, given all of the relevant source image pixels. Again, the network does not just perform, e.g., a simple average, but learns to optimally combine the source pixels using training data.

The two towers in our network correspond to these two tasks: the selection tower produces a probability map (or *selection map*) for each depth indicating the likelihood of each pixel having that depth. The color tower produces a full color output image for each depth; one can think of this tower as producing the best color it can for each depth, assuming that the depth is the correct one. These D color images are then combined as a per-pixel weighted sum with weights drawn from the selection maps: the selection maps decide on the best color layers to use for each output pixel. This simple new approach to view synthesis has several attractive properties. For instance, we can learn all of the parameters of both towers simultaneously, end-to-end using deep learning methods. The weighted averaging across color layers also yields some resilience to uncertainty—regions where the algorithm is not confident tend to be blurred out, rather than being filled with warped or distorted input pixels.

More formally, the selection tower computes, for each pixel $p_{i,j}$, in each plane P_z, the *selection probability* $s_{i,j,z}$ for the pixel being at that depth. The color tower computes for each pixel $p_{i,j}$ in each plane P_z the color $c_{i,j,z}$ for the pixel at that plane. The final output color for each pixel is computed as a weighted summation over the output color planes, weighted by the selection probability (Fig. 3):

$$c^f_{i,j} = \sum s_{i,j,z}\, c_{i,j,z}. \qquad (1)$$

The input to each tower is the set of plane sweep volumes V^k_C (consisting of $N \times D$ reprojected images in total over all volumes, where N is the number of source images, and D is the number of depth planes). The first layer of each tower operates on each reprojected image P^i_k independently, allowing it to learn low-level image features. After the first layer, the feature maps corresponding to the N sources are concatenated per depth plane, and subsequent layers operate on these per-depth-plane feature maps. The final layers of the selection tower additionally use connections across depth planes.

The selection tower. The selection tower (Fig. 4) consists of two main stages. The early layers, as discussed, consist of a number of $2D$ convolutional rectified linear layers that share weights across all depth planes (and within a depth plane for the first layer.) Intuitively, the early layers will compute features that are independent of depth, such as pixel differences, so their weights can be shared. The final set of layers are connected across depth planes in order to model interactions between depth planes, such as those caused by occlusion (e.g., the network might learn to prefer closer planes that have high scores in case of ambiguities in depth). The final layer of the network is a per-pixel *softmax* normalization transformer over depth. The *softmax* transformer encourages the model to pick a single depth plane per pixel, whilst ensuring that the sum over all depth planes is 1. We found that using a $\tanh$ activation for the penultimate layer gives more stable training than the more natural choice of a linear layer. In our experiments the linear layer would often "shut down" certain depth planes[1] and never recover, presumably due to large gradients from the *softmax* layer. The output of the selection tower is a 3D volume of single-channel nodes $s_{i,j,z}$ where $\sum_{z=1}^{D} s_{i,j,z} = 1$.

The color tower. The color tower (Fig. 5) is simpler and consists of only 2D convolutional rectified linear layers that

[1]The depth planes would receive zero weight for all inputs and pixels.

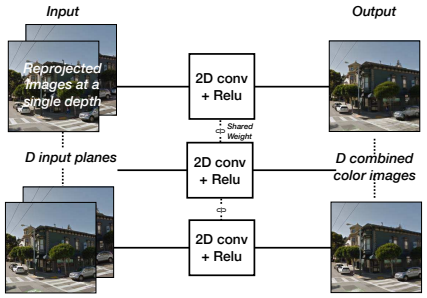

Figure 5: The color tower learns to combine and warp pixels across sources to produce a color $c_{i,j,z}$ for each pixel $p_{i,j}$ in each depth plane P_z. As in the selection tower, the first 2D layer operates on the individual reprojected images. Subsequent layers operate on the concatenated features per depth plane.

share weights across all planes, followed by a linear reconstruction layer. Occlusion effects are not relevant for the color layer so no across-depth interaction is needed. The output of the color tower is again a 3D volume of nodes $c_{i,j,z}$. Each node in the output has three channels (RGB).

The output of the color tower and the selection tower are multiplied together per node to produce the output image c^f (Eq. 1). During training the resulting image is compared with the known target image I^t using a per-pixel L_1 loss. The total loss is thus: $L = \sum_{i,j} |c^t_{i,j} - c^f_{i,j}|$ where $c^t_{i,j}$ is the target color at pixel i, j.

Multi-resolution patches. Rather than predict a full image at a time, we predict the output image patch-by-patch. We found that passing in a set of lower resolution versions of successively larger areas around the input patches improved results by providing the network with more context. We pass in four different resolutions. Each resolution is first processed independently by several layers and then upsampled and concatenated before entering the final layers. The upsampling uses nearest neighbor interpolation. The full details of the complete network are shown in Fig. 6.

Training. Training our network is a matter of simply taking a posed set of images, leaving one image out, and predicting it from the remaining ones. To evaluate the effect of different types of training data, we trained two networks from two distinct image sources. The first network used images from Street View. These images were posed using a combination of odometry and traditional structure-from-motion methods [18]. The vehicle captures a set of images, known as a rosette, from different directions for each exposure. Each camera uses a rolling shutter sensor, which is taken into account by our camera model. We used approximately 100K of such image sets during training. The second network was trained using the posed image sequences (00-10) from from the KITTI odometry dataset [12]. During training, we held out sequence 04 as a validation set for hyperparameter training, as well as sequence 10 for use as a test set for final evaluation; we trained on the remaining approximately 20k images. We used both of the color stereo cameras during training and testing.

Our network uses subsets of $N + 1$ images during training (where we used $N = 4$ for all experiments in this paper). The center image is used as the target and the other N are used as input. We used a continuously running online sample generation pipeline that selected a random subset from the training imagery and reprojected random patches from this subset. The network was trained to produce 8×8 patches from overlapping input patches of size 30×30. We used $D = 96$ depth planes in all results shown. Since the network is fully convolutional, there are no border effects as we transition between patches in the output image. In order to increase the variability of the patches during training, patches from many images are mixed together to create mini-batches of size 96. We trained our network using Adagrad [8] with an initial learning rate of 0.0001 using the system of Dean *et al*. [5]. In our experiments, training converged after approximately 1M steps. Due to sample randomization, it is unlikely that any patch was used more than once in training.

4. Results

To evaluate our model quantitatively on the task of view interpolation, we generated novel images from the same viewpoint as known (but withheld) images from our KITTI test set, consisting of 200 randomly selected images from sequence 10, camera 2. We then measured the L_1 per-pixel, per-channel, prediction error of the synthesized vs. ground truth image. To evaluate the effect of varying the camera spacing (i.e. baseline) of the source images on our method, we tried several experiments where we varied the baseline of the images used during training. The median distance between the original KITTI images is 0.8m. To generate different baselines, we used variants of the dataset where we took every other (median spacing 1.6m), or every third image (median spacing 2.4m), creating natural variations in baselines. We used the same set of held-out images independently of baseline—only the source input images used for view interpolation changed. We show the prediction error on a random sample of 200 images from the training data. As expected this is comparable but slightly lower than the test error. We show quantitative results in Table 1, and qualitative results in 8.

All three KITTI-trained models did well on the natural baseline (0.8m) and the wide baseline (1.6m) test data. They all had difficulty on the very wide baseline (2.4m) test data. All images shown used the wide baseline trained model. The size of baselines that the models were trained on did

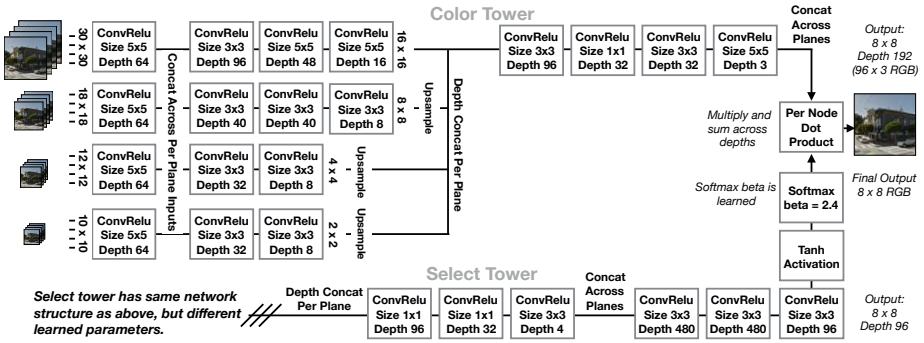

Figure 6: **Full network diagram.** The initial stages of both the color and selection towers are the same structure, but do not share parameters.

	KITTI L_1 Prediction Error		
	Test: 0.8m	Test: 1.6m	Test: 2.4m
KITTI training error	7.0	10.23	13.53
Trained on KITTI (0.8m)	7.49	10.41	13.44
Trained on KITTI (1.6m)	7.60	10.28	12.97
Trained on KITTI (2.4m)	8.06	10.73	13.37
Trained on Street View	10.50	15.49	20.40
Depth Splatting	10.80	15.92	20.08

Table 1: **Prediction error for varying training and test sets.** Results are shown as L_1 image prediction error on the test set (lower is better). The first row of numbers shows the training error of our method on each spacing. Each subsequent row shows test set errors for each training set and method: KITTI training sets with 0.8m, 1.6m, and 2.4m spacings, the Street View training set, and the depth splatting baseline approach. Each column shows prediction scores for KITTI test sets with different spacings.

not appear to have a large effect. As with other methods, our method had difficulty primarily near the left and right edges of the images, e.g. last row of 8, where the relative pixel motion is very fast. Additionally it occasionally had problems with thin structures, but less often than the competing methods, e.g. second last row of 8.

The Street-View-trained model did not do as well on this dataset. During training, every Street View rosette provides pixels in every direction, so the reprojected depth planes always have valid pixels. With the KITTI dataset, however, some parts of the depth planes were not visible from all cameras. Since the model had not seen such missing pixels during training, it made large errors. This chiefly affects the boundaries of the rendered images; the interiors where all sources are available look much better. It is likely that a training regime that randomly removed parts of the Street View images would mitigate this problem.

Comparison to prior work. For comparison, we implemented a baseline IBR algorithm that computes depth using [29] for the four nearest input images and splats the pixels from the two nearest images into the target view. To fill any small remaining holes, we diffused neighboring valid pixels. We ran this algorithm on the KITTI test set and computed L_1 pixel differences as before (Table 1, last row). Our method (trained on KITTI) outperformed the simple IBR algorithm on all spacings. We note that these scenes are difficult for stereo algorithms because of the abundance of natural vegetation, specularities and thin structures such as lampposts.

Chaurasia *et al*. [1] attempted to run their algorithms on the same KITTI data, but their algorithms were unable to generate images on this data. They mentioned difficulty obtaining a good initial reconstruction due to challenging lighting conditions, vegatation and high image noise. As we discuss below, their algorithm works well on other types of data.

Additionally, we compare our method to one that uses a recent state-of-the-art optical flow algorithm [26] to interpolate an in-between image. There is no notion of 3D pose when doing optical flow, so the interpolated image is only approximately at the viewpoint of the witheld image. Additionally [26] uses only two images as input so the comparison is not completely fair. However even in interior image areas our method looks qualitatively better, e.g. the third row of 8.

Additional images, as well as an interpolated video, are included as supplemental material.

Additional datasets. Finally, we used our method to interpolate from images featured in the work of Chaurasia, *et al*. [2, 1]. The results are shown in Figure 9. We again held out a known image and predicted from nearby images. The authors provided us with the result of their algorithm on these held-out images, and we compared our results using both the KITTI-trained model and the Street-View-trained model. The images in this dataset were captured with a handheld DSLR camera, and hence are quite different from

our training images, particularly from the KITTI data. Despite the fact that our models were not trained directly for this task, they did a reasonable job at reproducing the input images. Our method also had difficultly with the repeating structure and occluding pillars in the second example; the Chaurasia method likely benefited from being able to use more images to resolve this ambiguity. As in the KITTI examples, the Street-View-trained model again had difficulty when some depth planes were missing pixels, as can be seen seen near image boundaries.

Overall, our model produces plausible outputs that are difficult to immediately distinguish from the original imagery. Our model can handle a variety of traditionally difficult surfaces, including trees and glass as shown in Figure 1. Although the network does not attempt to model specular surfaces, the images in Figure 8 show that performance degrades gracefully in their presence. Noticeable artifacts in our results include a slight loss of resolution and the occasional disappearance of thin foreground structures. Moving objects, which occur often in training, are handled gracefully by our model: they are blurred in a manner that evokes motion blur (e.g. the flag in Figure 9). On the other hand, violating the maximum camera motion assumption significantly degrades the quality of the interpolated results. While our network can theoretrically model occlussion, we find that it does not always perform correctly and that thin objects occasionally fade into the background. This can be seen at the top of the lamppost in the fourth row of Figure 8.

In order to better demonstrate what the network learns, we have included crops of the the input reprojected images and layer outputs from the color and selection towers for a single depth plane in Figure 7. The particular depth planes shown have been chosen so that the cropped regions have strong selection probability at that plane, as shown in the selection layer output. As shown in that figure, the color layer does more than simply average the input reprojected images: it learns to warp and robustly combine the input to produce the color image for that depth plane. This ability allows us to have depth planes that are separated by more than one pixel of disparity.

Computational Costs. The network renders images in small patches, as rendering an entire image would be prohibitively expensive in RAM, and takes about 12 minutes on a multi-core workstation to render a 512×512 pixel image. If the convolutional nature of the network were fully exploited, approximately 15 TFlops (multiply-adds) would be needed to render the same image.

5. Discussion

We have shown that it is possible to train a deep network end-to-end to perform novel view synthesis. Our method is versatile and requires only sets of posed imagery. Results comparing real views with synthesized views show the generality of our approach. Our results are competitive with existing image-based rendering methods, even though our training data is considerably different from the test sets.

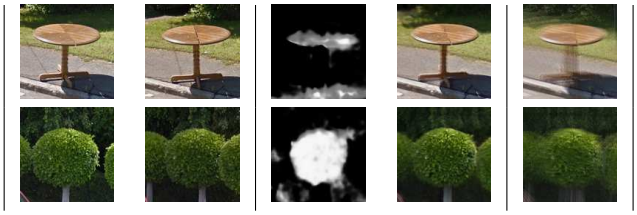

Figure 7: Layer outputs at one depth plane, showing reprojected input views (left), the outputs of the selection and color layers at a given depth (middle), and a comparison to the average (right). Note that while averaging produces ghosting, the color layer can tolerate misalignment.

One drawback of our method is speed, which is minutes per image. However, based on an analysis of the total FLOPs required, we believe that an optimized GPU implementation could render a 512×512 image in just a few seconds. We have not performed extensive experiments on the depths of the internal network layers, reducing these (while maintaining quality) may offer large speed-ups. Separable convolutions [16] and quantization [36] could also be used to speed up inference.

Our method currently requires reprojecting each input image to a set of depth planes; we currently use 96 depth planes, which limits the resolution of the output images that we can produce. Increasing resolution would require a larger number of depth planes, which would mean that the network takes longer to train and run. This is a drawback shared with other volumetric stereo methods. However, our method requires reprojected images per rendered frame, rather than just once when creating the scene. We plan to explore pre-computing parts of the network and warping to new views before running the final layers.

Another interesting direction of future work is to explore different network architectures. For instance, one could use a recurrent network to process the reprojected depth images one depth plane at a time. Such a network would not need expensive connections across depth. We believe that, with some of these improvements, our method could offer real-time performance on a GPU.

Our network is trained using four input views per target view. We currently cannot change the number of input views after training, which is suboptimal when there are denser sets of cameras that can be exploited, as in sequences from [1]. One idea is to choose the set of input views per pixel; however, this risks introducing discontinuties at transitions between chosen views. Alternatively, a more complex recurrent model could handle arbitrary numbers of input views, though this would likely complicate training. It

Figure 8: Comparison of real to synthesized KITTI images, showing results from the Kitti- and Street-View-trained model, a depth splatting algorithm from computed depth maps [29] and an optical flow algorithm [26]. The optical flow interpolates images halfway between the two input images.

Figure 9: Comparison of real to synthesized images from Chaurasia [1], showing results from the Kitti- and Street-View-trained model with those from Chaurasia [1].

would also be interesting to investigate using the outputs of the internal layers of the network. For instance, it is likely that the network learns a strong pixel similarity measure in the select tower that could be incorporated into a more traditional stereo framework.

Finally, similar networks could likely be applied to other problems, such as synthesizing intermediate frames in video or predicting a depth map, given appropriate training data.

References

[1] G. Chaurasia, S. Duchêne, O. Sorkine-Hornung, and G. Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics*, 32, 2013. 1, 2, 6, 7, 8

[2] G. Chaurasia, O. Sorkine, and G. Drettakis. Silhouette-aware warping for image-based rendering. *Computer Graphics Forum (Proc. EGSR)*, 30(4), 2011. 2, 6

[3] R. T. Collins. A space-sweep approach to true multi-image matching. In *CVPR*, 1996. 3

[4] A. Criminisi, A. Blake, C. Rother, J. Shotton, and P. Torr. Efficient dense stereo with occlusions for new view-synthesis by four-state dynamic programming. 2007. 1

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *NIPS*, pages 1232–1240. 2012. 5

[6] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *ECCV*. 2014. 2

[7] A. Dosovitskiy, J. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 1, 3

[8] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010. 5

[9] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. 2

[10] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating textures. *Computer Graphics Forum (Proc. of Eurographics EG)*, 27(2):409–418, Apr. 2008. Received the Best Student Paper Award at Eurographics 2008. 2

[11] A. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. In *ICCV*, 2003. 2

[12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 1, 2, 5

[13] M. Goesele, J. Ackermann, S. Fuhrmann, C. Haubold, R. Klowsky, D. Steedly, and R. Szeliski. Ambient point clouds for view interpolation. In *SIGGRAPH*, 2010. 2

[14] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH*, 1996. 2

[15] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. In *SIGGRAPH*, 2005. 2

[16] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014. 7

[17] K. Karsch, C. Liu, and S. B. Kang. DepthTransfer: Depth extraction from video using non-parametric sampling. *PAMI*, 2014. 2

[18] B. Klingner, D. Martin, and J. Roseborough. Street view motion-from-structure-from-motion. In *ICCV*, December 2013. 2, 5

[19] K. R. Konda and R. Memisevic. Unsupervised learning of depth and motion. *CoRR*, abs/1312.3429, 2013. 2

[20] J. Konrad, M. Wang, and P. Ishwar. 2d-to-3d image conversion by learning depth from examples. In *CVPR Workshops*, 2012. 2

[21] J. Kopf, M. F. Cohen, and R. Szeliski. First-person hyperlapse videos. In *SIGGRAPH*, 2014. 1

[22] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2

[23] T. Kulkarni, W. Whitney, K. Pushmeet, and J. Tenenbaum. Deep convolutional inverse graphics network. *arXiv preprint arXiv:1503.03167*, 2015. 1, 2

[24] T. D. Kulkarni, W. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. *CoRR*, abs/1503.03167, 2015. 3

[25] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH*, 1996. 2

[26] C. Liu. *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis*. PhD thesis, Massachusetts Institute of Technology, 2009. 6, 8

[27] R. Memisevic and C. Conrad. Stereopsis via deep learning. In *NIPS workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 2

[28] G. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014. 2

[29] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, June 2011. 6, 8

[30] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *PAMI*, 31(5), May 2009. 2

[31] S. M. Seitz and C. R. Dyer. View morphing. In *SIGGRAPH*, 1996. 2

[32] Q. Shan, R. Adams, B. Curless, Y. Furukawa, and S. M. Seitz. The visual turing test for scene reconstruction. In *Proc. 3D Vision*, 2013. 2

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 2

[34] R. Szeliski. Prediction error as a quality metric for motion and stereo. In *ICCV*, 1999. 2

[35] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *IJCV*, 32(1), 1999. 3

[36] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011. 7

[37] S. Vedula, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. *PAMI*, 27(3), March 2005. 2

[38] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. *CoRR*, abs/1411.4958, 2014. 2

[39] O. Woodford, I. Reid, P. Torr, and A. Fitzgibbon. On new view synthesis using multiview stereo. 2007. 1

[40] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *NIPS*, 2012. 2

[41] K. Yamaguchi, T. Hazan, D. A. McAllester, and R. Urtasun. Continuous markov random fields for robust stereo estimation. *CoRR*, abs/1204.1393, 2012. 2

[42] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. June 2015. 2

[43] L. Zhang and S. M. Seitz. Estimating optimal parameters for mrf stereo from a single image pair. *PAMI*, 29(2), February 2007. 2

[44] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. In *SIGGRAPH*, 2004. 2