Revealing Disocclusions in Temporal View Synthesis through Infilling Vector Prediction
Supplementary Material

Vijayalakshmi Kanchana Nagabhushan Somraj Suraj Yadwad Rajiv Soundararajan
Indian Institute of Science, Bengaluru, India
{vijayalaksh1, nagabhushans, surajyadwad, rajivs}@iisc.ac.in

Overview

The contents of this supplementary material has been organized into the following sections:

A. Video samples
B. Pose warping
C. Temporal prior estimation
D. Details of the network architecture
E. Example scenes from our database
F. Additional qualitative results

A. Video Samples

We show results of our model on a longer video sequence of 5 seconds from our database and compare it with EdgeConnect [4] and Synsin [8]. The videos are attached along with this pdf. In the EdgeConnect video, we see flicker in the infilled regions next the cubes in the video. One such region is highlighted with a bounding box in the video. This is due to incorrect infilling by EdgeConnect, which appears as a flicker since every alternate frame is graphically rendered. In the SynSin video, we again see flicker artifacts, perhaps more dominant in the known region. As explained in Section B.2, this is perhaps due to incorrect decoding of features to RGB intensity values in the known regions by SynSin. In both cases, our model is able to produce better videos with minimal artifacts.

B. Pose Warping

B.1. Our Method

In this section, we elaborate on the warping equation (1) in the main paper and explain the interpolation/splatting we employ. We represent a point (i, j) in frame f_n as $p_n(i, j)$. Let $\hat{p}_n(i, j) = [i, j, 1]^T$ represent the point location in homogeneous coordinates. We first unproject $\hat{p}_n(i, j)$ using depth d_n and the 3×3 camera intrinsic matrix K, to find the 3d point coordinates in view n

$$P_n(i, j) = d_n(i, j)\, K^{-1}\, \hat{p}_n(i, j) \tag{5}$$

We then convert $P_n(i, j)$ to homogeneous coordinates by appending 1 in the fourth dimension to get $\hat{P}_n(i, j)$. Using the relative transformation from frame f_n to frame f_{n+1} as T_n, we find the 3d point coordinates in view $n + 1$ in homogeneous form as

$$\hat{P}_{n+1}(i, j) = T_n \hat{P}_n(i, j) \tag{6}$$

By removing the fourth dimension of $\hat{P}_{n+1}(i,j)$, we get the 3d coordinates $P_{n+1}(i,j)$. Note that, $P_{n+1}(i,j)$ provides 3d coordinates in view $n+1$ corresponding to the point $p_n(i,j)$ in frame f_n. The third coordinate of $P_{n+1}(i,j)$ gives the depth in view $n+1$ for the point $p_n(i,j)$ and is denoted as $d^w(i,j)$. We then project the point $P_{n+1}(i,j)$ to the image plane of view $n+1$ by multiplying with camera matrix to find the location of the point $p_n(i,j)$ in frame f_{n+1} in homogeneous form

$$\hat{p}_{n+1}(i,j) = K\ P_{n+1}(i,j) \tag{7}$$

By normalizing the 3rd dimension to 1, we get the actual 2d coordinates as $p_{n+1}(i,j)$. Thus, we obtain the 2d coordinates $p_{n+1}(i,j)$ in frame f_{n+1} corresponding to the point $p_n(i,j)$ in frame f_n. Let $(p^x_{n+1}(i,j), p^y_{n+1}(i,j))$ represent the x and y components of $p_{n+1}(i,j)$. We copy the intensity from frame f_n to frame f_{n+1} as

$$f_{n+1}(p^x_{n+1}(i,j), p^y_{n+1}(i,j)) = f_n(i,j) \tag{8}$$

We repeat the above for every pixel $p_n(i,j)$ in frame f_n, where $i = 1, 2, \ldots, W$ and $j = 1, 2, \ldots, H$. W and H are the width and height of the frame respectively.

However, the locations $p_{n+1}(i,j)$ can be fractional and hence to reconstruct the known regions of frame f_{n+1}, we need to interpolate the intensities at the grid locations. To obtain the intensities at the grid locations, we use inverse bilinear interpolation employed in [7]. In bilinear interpolation, the intensity at an intermediate location within a grid is computed as a weighted combination of the intensities at nearest grid locations. Here, we do the opposite i.e. given the intensities at intermediate locations, we compute the intensities at grid locations. Hence, we term this operation as inverse bilinear interpolation.

We compute the intensity at pixel (m,n) in frame f_{n+1} as follows. We consider a 2×2 neighborhood $N(m,n)$ around the pixel (m,n) and compute weighted average of the points within this grid.

$$f^w_{n+1}(m,n) = \frac{\sum_{(i,j)\in N(m,n)} w(m,n,i,j)\ f_n(i,j)}{\sum_{(i,j)\in N(m,n)} w(m,n,i,j)} \tag{9}$$

We use two sets of weights, one based on proximity w_p and the other based on depth w_d.

$$w(m,n,i,j) = w_p(m,n,i,j)\ w_d(i,j) \tag{10}$$

The proximity weights are computed similar to bilinear interpolation as

$$w_p(m,n,i,j) = (1 - |m - p^x_{n+1}(i,j)|) \cdot (1 - |n - p^y_{n+1}(i,j)|) \tag{11}$$

Using depth weights, we provide higher weightage to foreground objects (lower depth) and lower weightage to background objects (higher depth). As a result, when there is an occlusion i.e. multiple objects map to a same location in frame f_{n+1}, the intensity of the object nearest to camera is picked and not that of occluded objects. We compute the depth weights as

$$w_d(i,j) = \frac{1}{(1 + d^w(i,j))^\gamma} \tag{12}$$

Recall that $d^w(i,j)$ is obtained as the third coordinate of $P_{n+1}(i,j)$ obtained in Equation 6. Here, γ is a global scaling constant that widens the gap between weights of foreground and background objects. By trial and error on 5 scenes, we found γ that gives best reconstruction as

$$\gamma = \frac{50}{\max\limits_{1\le k\le W, 1\le l\le H} \log(1 + d^w(k,l))} \tag{13}$$

Also, since outdoor scenes can have very large depth values (eg. sky), we clip d^w to the range $[0, 1000]$.

Model	RVS	StereoMag	SynSin	3DP	Ours
K-MSE	52	201	114	72	42

Table 3: Comparison of different warping methods in terms of error in known regions.

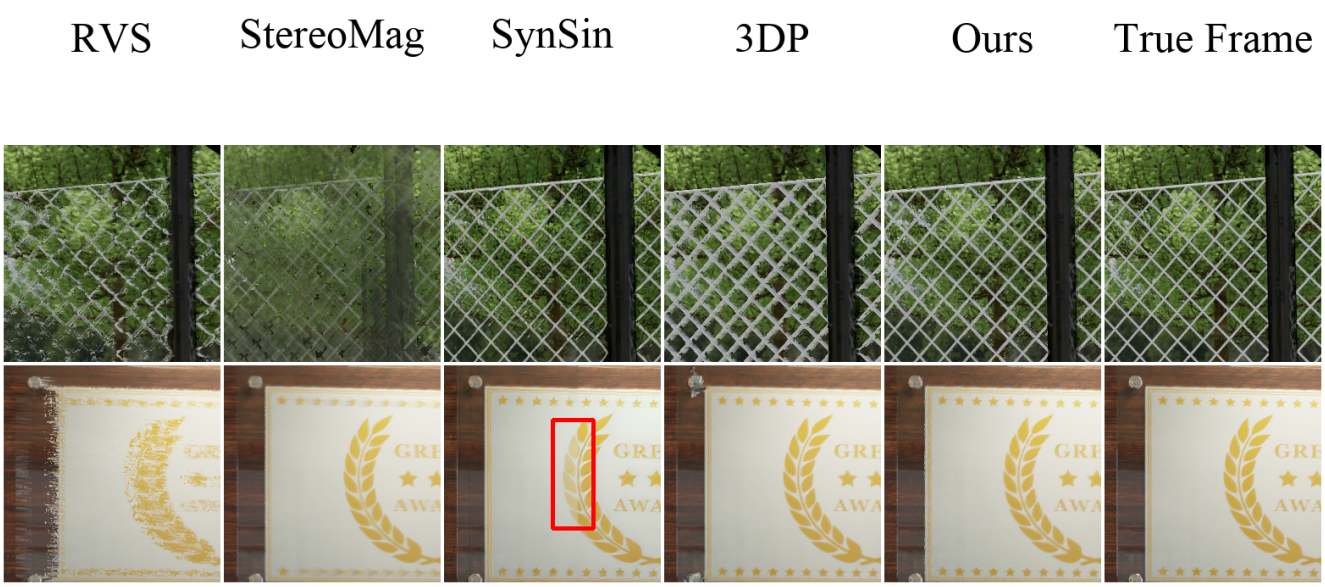

Figure 8: Comparison of known region reconstruction by different methods. Disoccluded regions are filled with ground truth values.

B.2. Comparison with Other Methods

We compare our warping method with the other methods used in the view synthesis algorithms based on reconstruction of known regions in our test set. The Reference View Synthesizer (RVS) [2] from the Motion Pictures Experts Group (MPEG) uses mesh based warping. It divides each grid into two triangles and warps the triangles. As a result, in the disoccluded regions, the triangles get stretched which leads to artifacts as seen in Figure 11. StereoMag [9] and 3DP [6] build a representation of the scene from the input and then generate the frame for the given viewpoint. While StereoMag builds Multi-Plane Images (MPI) from two past frames, 3DP builds Layered Depth Image (LDI) from the RGB-D input, as the internal scene representation. SynSin [8] is the closest to our approach. They warp the points using depth and then use learnable splatting module provided by PyTorch3D [1] library. However, they warp the points in feature space rather than in RGB space. We note that, only our approach and RVS are not learning based, while the other three use learning components to reconstruct the known region. Also, StereoMag is the only approach that does not use ground truth depth and uses two past frames, while all other approaches use a single frame and its ground truth depth.

In Table 3, we compare the different methods by computing mean squared error in the known regions (K-MSE). We show few samples from all the approaches in Figure 8. We observe that our approach has least MSE and also has minimum visual artifacts. While RVS is a close contender in terms of MSE, it is more susceptible to small errors in depth map, which are not uncommon in graphically rendered videos. Since SynSin warps the frame in feature space and then has to reconstruct intensities from the features, it sometimes fails to reconstruct color correctly, which can be seen in the second example. We also note that this shift in color leads to flicker, which can

be noticed in the attached video. For the first example, we observe that 3DP incorrectly increases the thickness of the fence. Since StereoMag does not use ground truth depth, its reconstructions can be blurry, as seen in first example.

C. Temporal Prior Estimation

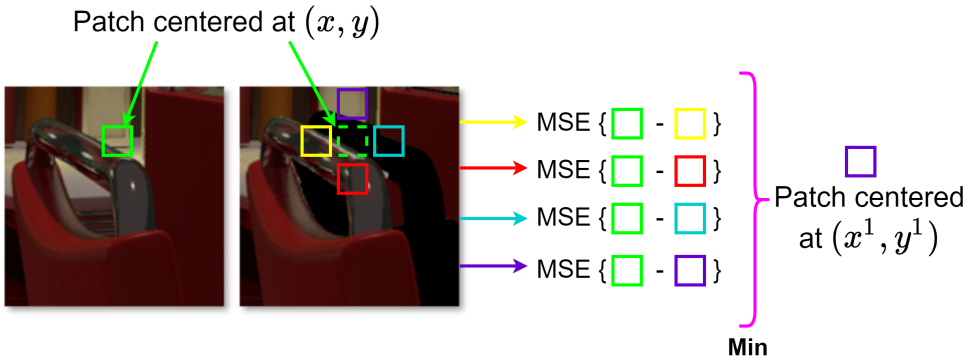

Figure 9: This figure illustrates the infilling vector estimation from the past frames, using the ground truth frame f_n and the frame f_n^w warped from f_{n-2}.

Extending from Section 4.2 in main paper, we illustrate the estimation of infilling vectors in frame f_n in Figure 9. For easy reference, we reproduce the textual explanation here.

"For every disoccluded pixel at (x, y) in the warped frame f_n^w, we search for the nearest neighbor in the known regions of the frame, in the four cardinal directions. We compare the intensities at these four locations with the true intensity at (x, y), available from the ground truth frame f_n. We pick the neighbor that has the least mean squared error (MSE) with the ground truth value, and set the infilling vector at (x, y) to point to this optimal neighbor's location. To avoid noisy estimates, we consider a small patch around the pixels, instead of individual pixels, and compute the MSE between the patches."

In Figure 9, the patch with purple square at (x^1, y^1) is the best match for the patch at (x, y) in ground truth frame. Hence, we set the fictitious infilling vector as

$$(\alpha_n, \beta_n) = (x^1 - x, y^1 - y) \tag{14}$$

D. Details of the Network Architecture

For depth processing, we use a 4 layer convolutional network. The network details are shown in Table 4. For infilling vector prediction, we use a U-Net [5] based architecture, consisting of a total of 9 layers including 4 subsampling layers and skip connections. The network details are shown in Table 5.

E. Example Scenes from Our Database

The 200 scenes in our database consist of both indoor and outdoor scenes, and have no object motion. Since egomotion is the only source of motion in the scene, and hence disocclusions, the scenes were chosen such that they had foreground objects that can cause disocclusions on appropriate camera motion. Figure 10 shows four indoor and outdoor scenes from the database.

Layer No.	1	2	3	4
Kernel size	7	7	7	7
No. of output channels	16	32	16	1
Stride	2	2	0.5	0.5
Padding	3	3	3	3
Activation	ReLU	ReLU	ReLU	Sigmoid

Table 4: Architecture details of depth processing network. Fractional strides represent upsampling followed by the convolution layer.

Layer No.	1	2	3	4	5	6	7	8	9
Kernel size	7	5	3	3	3	3	3	3	3
No. of output channels	64	128	256	256	256	256	128	64	2
Skip Connection	-	-	-	-	-	4	3	2	1
Stride	1	2	2	2	2	0.5	0.5	0.5	0.5
Padding	3	2	1	1	1	1	1	1	1
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	Linear

Table 5: Architecture details of infilling vector prediction network. Fractional strides represent upsampling followed by skip connection and the convolution layer. Skip connection at l^{th} layer specifies the layer number whose output is concatenated with the input of l^{th} layer.

Figure 10: Some examples of indoor and outdoor scenes from our database.

F. Additional Qualitative Results

In this section we show more samples of predicted frames from our database as well as from SceneNet RGB-D [3], for various benchmarks. Figure 11 shows comparison between different view synthesis models. Figure 12 shows comparison between different inpainting models. For examples from our database, we show cropped patches so that the disocclusions are visible.

References

[1] Accelerating 3d deep learning with PyTorch3D. *arXiv e-prints*, page arXiv:2007.08501, 2020.

[2] Sarah Fachada, Bart Kroon, Daniele Bonatto, Bart Sonneveldt, and Gauthier Lafruit. Reference view synthesizer (RVS) 2.0 manual, 2018.

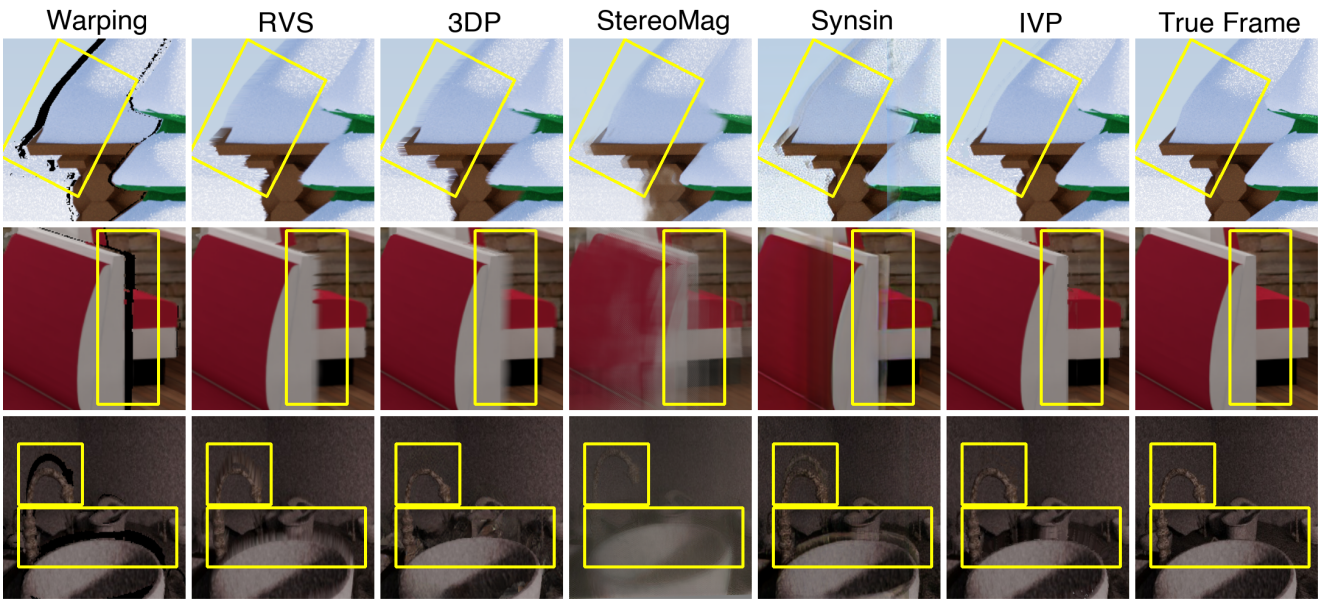

Figure 11: Comparison of temporally synthesized views by different view synthesis models. The first two rows correspond to samples from our database and the last row corresponds to a sample from SceneNet RGB-D database.

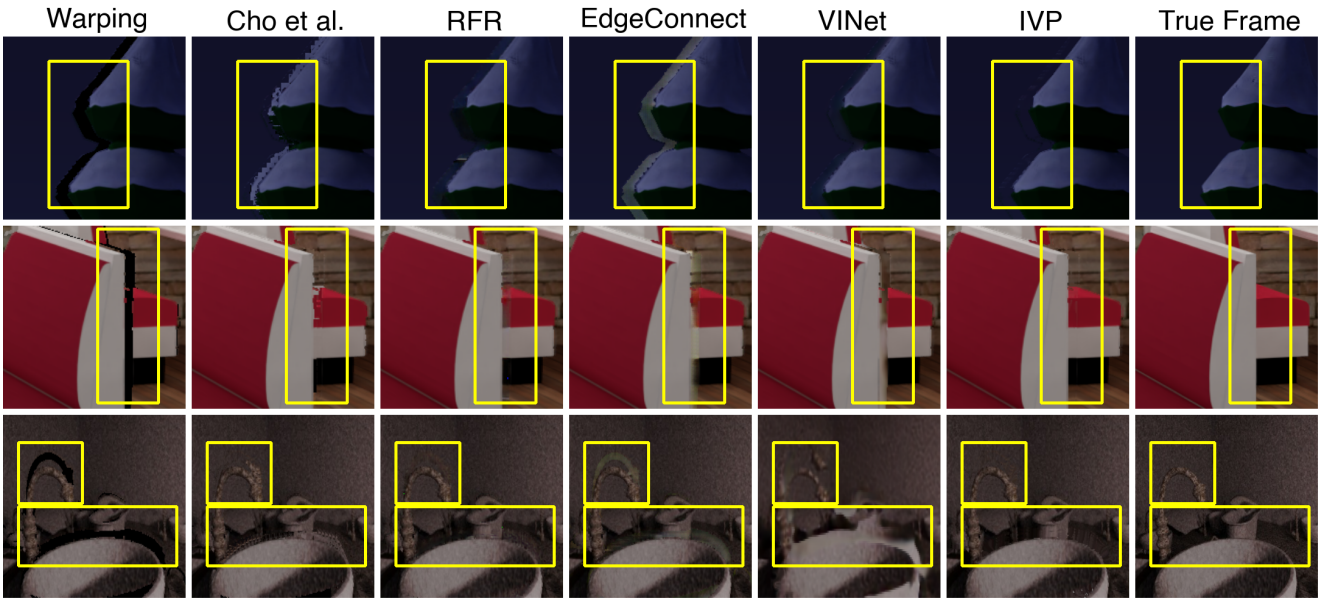

Figure 12: Comparison of temporally synthesized views by different inpainting models. The first two rows correspond to samples from our database and the last row corresponds to a sample from SceneNet RGB-D database.

[3] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2678–2687, 2017.

[4] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. Edgeconnect: Structure guided image inpainting using edge prediction. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct

2019.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[6] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8028–8038, 2020.

[7] Yang Wang, Yi Yang, Zhenheng Yang, Liang Zhao, Peng Wang, and Wei Xu. Occlusion aware unsupervised learning of optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4884–4893, 2018.

[8] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7467–7477, 2020.

[9] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018.