3D Moments from Near-Duplicate Photos —Supplementary Material—

Qianqian Wang[1,2] Zhengqi Li[1] David Salesin[1] Noah Snavely[1,2] Brian Curless[1,3] Janne Kontkanen[1]
[1]Google Research [2]Cornell Tech, Cornell University [3]University of Washington

1. LDI inpainting details

Given an RGBD image (where depth is obtained using the monocular depth estimator DPT [8]), we apply agglomerative clustering [5] in the disparity space to obtain its color LDI $\mathcal{L} \triangleq \{\mathbf{C}^l, \mathbf{D}^l\}_{l=1}^{L}$. We use the agglomerative clustering implementation from scikit-learn [7] and set the distance threshold to be 5, resulting in $2 \sim 5$ layers. We then inpaint the color and depth in each layer in a depth-aware manner.

To inpaint missing contents in layer l, we treat all the pixels between the l^{th} layer and the farthest layer as the context region (i.e., the region used as reference for inpainting). We exclude all foreground pixels in layers nearer than layer l because these layers may contain irrelevant pixels for inpainting layer l. To determine the region to be inpainted, we compute the smallest bounding box that contains all existing pixels in layer l, and pad this bounding box in all boundaries by 40 pixels. The missing pixels of the l^{th} layer within this bounding box is then set to be inpainted. We keep only inpainted pixels whose depths are smaller than the maximum depth of layer l so that inpainted regions do not mistakenly occlude layers farther than layer l.

When inpainting the color and depth of layer l, we erode the context pixels using a kernel of 3×3 to avoid unreliable depth predictions near the depth discontinuities as in [10]. Further, to avoid the inpainted regions in layer l to incorrectly occlude pixels in nearer layers, we clip the inpainted depth of each pixel in layer l to be slightly larger than the maximum depth of this pixel in the nearer layers.

2. Scene flow diffusion details

We perform a diffusion operation to complete the scene flow vectors in each LDI layer. Specifically, we first represent the scene flow layer as an optical flow layer and a depth layer. The optical flow layer can be obtained from the optical estimator directly. Each value in the depth layer is obtained by reprojecting its corresponding depth value in the other view into the current view. Note that only regions with mutual correspondences have valid values.

We then diffuse each optical flow layer and depth layer separately, so that inpainted and occluded pixels in the LDI also have defined scene flow vectors. Specifically, we apply a masked box filter of size 15×15 iteratively to each optical flow and depth layer to fill in the missing values. The mask in each iteration indicates current non-empty pixels that are used as reference for diffusion, and regions with mask value equal to 1 remain unchanged throughout the diffusion. Finally, we convert the completed optical flow and depth of each pixel to a scene flow vector, which will later be used to interpolate the point location at an intermediate time.

3. Network architecture

2D feature extractor. The 2D feature extractor generates a 32-dimensional feature map for each layer in the color LDI. We adapt ResNet34 [1] as implemented in PyTorch [6] to implement the feature extractor. The feature extractor is fully convolutional and accepts input images of variable size. We take a single image of size $640 \times 480 \times 3$ as an example input and present a detailed network architecture in Tab. 1.

Image synthesis network. We adopt a U-Net architecture for the image synthesis network, which takes in the rendered feature map and depth map to generate the final image. The network contains four down-sampling and up-sampling convolutional blocks with skip connections, where each convolutional block conducts a sequence of operations (convolution, GroupNorm [11], and ReLU) twice.

4. Training details

When sampling a triplet on Vimeo-90K [12] dataset, we randomly sample two frames that are 2 to 4 frames apart as the input frames, and a random frame between them as the ground truth. Similar idea applies to the MannequinChallenge [3] dataset, except that the interval between the two input frames ranges from 2 to 16. To train on MannequinChallenge [3] dataset, we align the monocular depth map with the SfM points by estimating a global scale and shift in the disparity space. We discard the triplet if 1) the aligned monocular depth maps have negative values, and 2) when we warp the input frame to the target frame, the percentage

Input (**id**: dimension)	Layer	Output (**id**: dimension)
0: $640 \times 480 \times 4$	7×7 Conv, 64, stride 2	**1**: $320 \times 240 \times 64$
1: $320 \times 240 \times 64$	3×3 MaxPool, stride 2	**2**: $160 \times 120 \times 64$
2: $160 \times 120 \times 64$	Residual Block 1	**3**: $160 \times 120 \times 64$
3: $160 \times 120 \times 64$	Residual Block 2	**4**: $80 \times 60 \times 128$
4: $80 \times 60 \times 128$	Residual Block 3	**5**: $40 \times 30 \times 256$
5: $40 \times 30 \times 256$	3×3 Upconv, 128, factor 2	**6**: $80 \times 60 \times 128$
[**4**, **6**]: $80 \times 60 \times 256$	3×3 Conv, 128	**7**: $80 \times 60 \times 128$
7: $80 \times 60 \times 128$	3×3 Upconv, 64, factor 2	**8**: $160 \times 120 \times 64$
[**3**, **8**]: $160 \times 120 \times 128$	3×3 Conv, 64	**9**: $160 \times 120 \times 64$
9: $160 \times 120 \times 64$	1×1 Conv, 32	**Out**: $160 \times 120 \times 32$

Table 1. **2D feature extractor network architecture**. 'Conv" represents a sequence of operations: convolution, rectified linear units (ReLU) and Batch Normalization [2]. "Upconv" represents a bilinear upsampling with certain factor, followed by a "Conv" operation with stride 1. "$[\cdot, \cdot]$" stands for channel-wise concatenation of two feature maps. The residual blocks are the same as those in the original ResNet34 [1] design. The output 32-dimensional feature map will be upsampled to have the same resolution as the input using bilinear interpolation.

of non-empty pixels in the target image space is less than 75% (this can either mean the camera motion is too large, or the monocular depth and SfM do not align).

5. Evaluation details

We describe the details of the experimental setup on the Nvidia [13] and UCSD [4] dataset. For the Nvidia dataset [13], we use the undistorted multi-view video sequences for evaluation. To form triplets, we first divide the multi-view videos into multiple segments, where each segment contains images at three consecutive time steps from all 12 camera viewpoints, resulting in 36 images in total. We run COLMAP [9] (masking out dynamic regions) to obtain the camera parameters and SfM point clouds for each segment. The UCSD dataset [4] instead provides the precomputed camera parameters from COLMAP [9]. However we still need to obtain the SfM points in order to align the monocular depth maps. We therefore perform triangulation using the views at a specific time step (masking out dynamic regions) to obtain the scene structure of static regions. Since their camera poses are fixed throughout each video sequence, we only need to perform the triangulation once for each multi-view video sequence.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 2

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 2

[3] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *CVPR*, pages 4521–4530, 2019. 1

[4] Kai-En Lin, Lei Xiao, Feng Liu, Guowei Yang, and Ravi Ramamoorthi. Deep 3d mask volume for view synthesis of dynamic scenes. *ArXiv*, abs/2108.13408, 2021. 2

[5] Oded Maimon and Lior Rokach. *Data Mining And Knowledge Discovery Handbook*. 2005. 1

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS-W*, 2017. 1

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 1

[8] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, 2021. 1

[9] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, pages 4104–4113, 2016. 2

[10] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *CVPR*, pages 8028–8038, 2020. 1

[11] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 1

[12] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *IJCV*, 127(8):1106–1125, 2019. 1

[13] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *CVPR*, pages 5336–5345, 2020. 2