Video Stitching for Linear Camera Arrays Supplementary Material

Wei-Sheng Lai[1,2]
wlai24@ucmerced.edu
Orazio Gallo[2]
ogallo@nvidia.com
Jinwei Gu[2]
gujinwei@gmail.com
Deqing Sun[2]
deqing.sun@gmail.com
Ming-Hsuan Yang[1]
mhyang@ucmerced.edu
Jan Kautz[2]
jkautz@nvidia.com

[1] University of California, Merced
[2] NVIDIA

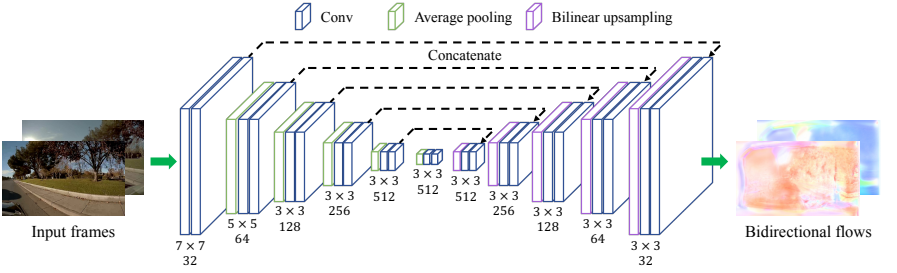
1 Overview

In this supplementary document, we present additional results to complement the paper. First, we provide the implementation and training details of the proposed method. Second, we analyze the robustness of the proposed method on the variation of the camera settings. Finally, we show some failure cases of our model. The video results are available in our project website at http://vllab.ucmerced.edu/wlai24/video_stitching/.

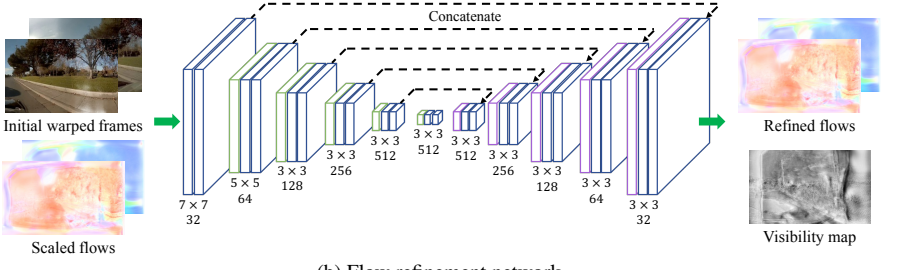
2 Implementation Details

Figure 1 shows the network architecture of our flow estimation and flow refinement networks. Both the flow estimation and flow refinement networks use the same U-Net architecture. All the convolutional layers are followed by the leaky ReLU activation layer with a negative slope of 0.1. In the encoder, we use the 2×2 average pooling layer to downsample feature maps by $2\times$. In the decoder, we use the bilinear upsampling layer instead of the transposed convolutional layer to upsample feature maps. We also add skip connections from the encoder to the decoder.

We implement our model using PyTorch [3]. During the training stage, we sample three consecutive frames from the same video in each forward pass. We only compute the temporal loss for the center frame while computing the content and perceptual losses for all the sampled frames. We set the initial learning rate to 10^{-4} and decrease by a factor of 2 for every 20,000 iterations. In total, we optimize our network using the ADAM solver [1] for 100,000 iterations We train our model on an NVIDIA Titan X GPU, which takes about 1 day



(a) Flow estimation network



(b) Flow refinement network

Figure 1: **Network architecture.**

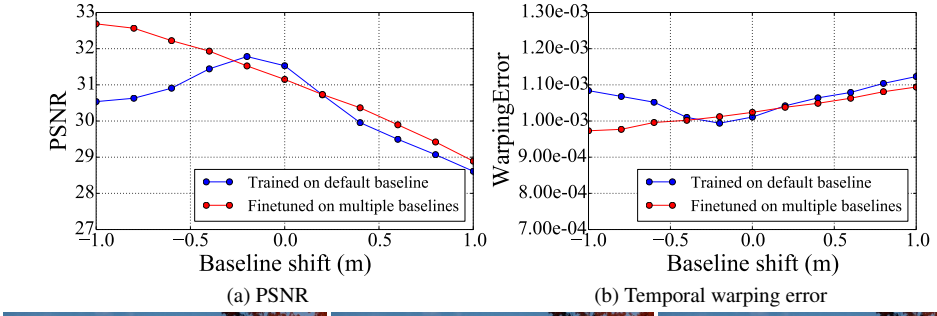
to converge. At the test stage, it takes 0.12 second for our network to stitch a video frame with a resolution of 1000×600 pixels.

3 Additional Analysis

Our method requires the cameras positions to be known, a fairly common assumption for algorithms that stitch the output of camera rigs. In fact, NVIDIA's VRWorks [2], against which we compare in the paper, also requires this information. Further, although trained for a specific configuration, our model is robust to deviations from the expected settings. To demonstrate the robustness of our model, we render the same test video by changing the camera baseline, *i.e.*, horizontally shifting the side cameras inward or outward from the position used for training.

Figure 2(a) and (b) offer an insight on the analysis of the impact of different baselines. Even when moving the side cameras inwards by up to 0.8m (62.5% of the original baseline) the PSNR drops by less than 1dB. While moving the side cameras outwards decreases the size of overlapped region, the PSNR drops less than 2dB when the deviation is up to 0.4m. On the other hand, the temporal warping error remains small and the video is still stable during playback. We show visual results from the default baseline and an extreme baseline shift (+1.0m) in Figure 2(c-d). Note that despite the very large change in baseline and associated reduction in image overlap when the baseline is increased, the quality remains similar—see the lamp post, which is in a transition region. However, minor artifacts can be seen in close-by regions, *e.g.*, double yellow lines.

We then fine-tune our model with data that mixes multiple camera baselines (from 0.28m to 2.28m). The fine-tuned model further improves the performance (red curves in Figure 2(a-b)) for a range of baseline shifts. In Figure 2(e), we show that our fine-tuned model success-



(a) PSNR (b) Temporal warping error

(c) 1.28m (default) (d) 2.28m (+1.0m) (e) 2.28m (+1.0m) fine-tuned

Figure 2: **Robustness analysis.** The model trained on a single camera setting is robust to a certain deviations from the default camera baseline. After fine-tuning the model on data with multiple camera baseline settings, the visual quality and temporal stability can be further improved.

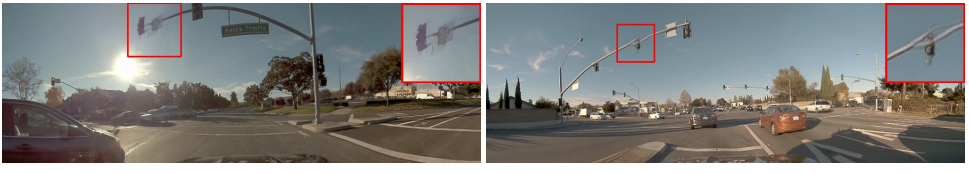


Figure 3: **Failure cases.** Our approach might produce mis-alignment or ghosting artifacts when the flow estimation is not accurate, especially for thin objects.

fully reduces the alignment artifacts in the double yellow lines. For dramatically different camera configurations or best quality, re-training or fine-tuning is a reasonable strategy, as the configuration is usually known before the deployment of the system.

4 Failure Cases

In Figure 3, we show two results where our model does not perform well. As the proposed pushbroom interpolation layer relies on the optical flow, the alignment may fail due to inaccurate flow estimation, especially on long and thin objects, e.g., utility poles or street lamps.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[2] Nvidia VRWorks. https://developer.nvidia.com/vrworks/vrworks-360video. 2

[3] 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. 2017. 1