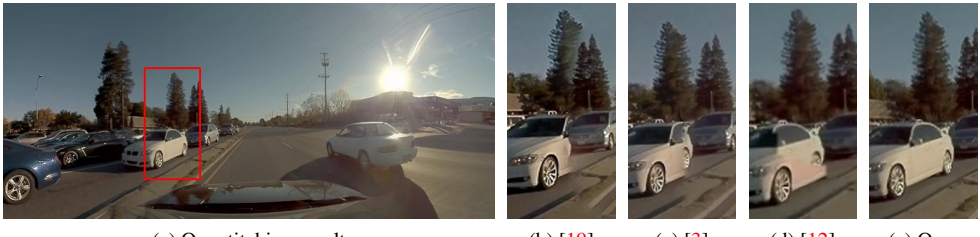
Video Stitching for Linear Camera Arrays

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

Abstract

Despite the long history of image and video stitching research, existing academic and commercial solutions still produce strong artifacts. In this work, we propose a wide-baseline video stitching algorithm for linear camera arrays that is temporally stable and tolerant to strong parallax. Our key insight is that stitching can be cast as a problem of learning a smooth spatial interpolation between the input videos. To solve this problem, inspired by pushbroom cameras, we introduce a fast pushbroom interpolation layer and propose a novel pushbroom stitching network, which learns a dense flow field to smoothly align the multiple input videos for spatial interpolation. Our approach outperforms the state-of-the-art by a significant margin, as we show with a user study, and has immediate applications in many areas such as virtual reality, immersive telepresence, autonomous driving, and video surveillance.



(a) Our stitching result (b) [19] (c) [3] (d) [12] (e) Ours

Figure 1: **Examples of video stitching.** Inspired by pushbroom cameras, we propose a deep pushbroom stitching network to stitch multiple wide-baseline videos of dynamic scenes into a single panoramic video. The proposed learning-based algorithm outperforms prior work with minimal mis-alignment artifacts (*e.g.*, ghosting and broken objects). More video results are presented in the supplementary material.

1 Introduction

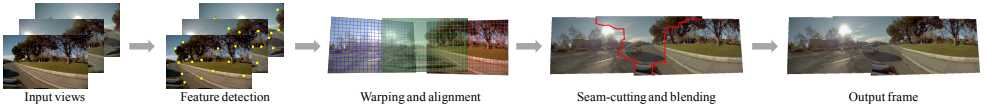
Due to sensor resolution and optics limitations, the field of view (FOV) of most cameras is too narrow for applications such as autonomous driving and virtual reality. A common solution is to stitch the outputs of multiple cameras into a panoramic video, effectively extending the FOV. When the optical centers of these cameras are nearly co-located, stitching can be solved with a simple homography transformation. However, in many applications, such as autonomous driving, remote video conference, and video surveillance, multiple cameras have to be placed with *wide baselines*, either to increase view coverage or due to some physical constraints. In these cases, even state-of-the-art methods [12, 21] and current commercial solutions (*e.g.*, VideoStitch Studio [27], AutoPano Video [3], and NVIDIA VRWorks [19]) struggle to produce artifact-free videos, as shown in Figure 1.

One main challenge for video stitching with wide baselines is *parallax*, *i.e.* the apparent displacement of an object in multiple input videos due to camera translation. Parallax varies with object depth, which makes it impossible to properly align objects without knowing dense 3D information. In addition, occlusions, dis-occlusions, and limited overlap between the FOVs also cause a significant amount of stitching artifacts. To obtain better alignment, existing image stitching algorithms perform content-aware local warping [6, 30] or find optimal seams around objects to mitigate artifacts at the transition from one view to the other [9, 31]. Applying these strategies to process a video frame-by-frame inevitably produces noticeable jittering or wobbling artifacts. On the other hand, algorithms that explicitly enforce temporal consistency, such as spatio-temporal mesh warping with a large-scale optimization [12], are computationally expensive. In fact, commercial video stitching software often adopts simple seam cutting and multi-band blending [5]. These methods, however, often cause severe artifacts, such as ghosting or misalignment, as shown in Figure 1. Moreover, seams can cause objects to be cut off or completely disappear from stitched images—a particularly dangerous outcome for use cases such as autonomous driving.

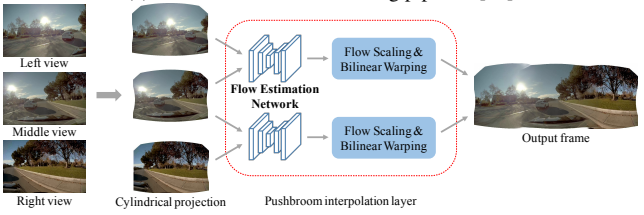
We propose a video stitching solution for linear cameras arrays that produces a panoramic video. We identify three desirable properties in the output video: (1) Artifacts, such as ghosting, should not appear. (2) Objects may be distorted, but should not be cut off or disappear in any frame. (3) The stitched video needs to be temporally stable. With these three desiderata in mind, we formulate video stitching as a spatial view interpolation problem. Specifically, we take inspiration from the pushbroom camera, which concatenates vertical image slices that are captured while the camera translates [10]. We propose a pushbroom stitching network based on deep convolutional neural networks (CNNs). Specifically, we first project the input views onto a common cylindrical surface. We then estimate bi-directional optical flow, with which we *simulate* a pushbroom camera by interpolating all the intermediate views between the input views. Instead of generating all the intermediate views (which requires multiple bilinear warping steps on the entire image), we develop a *pushbroom interpolation layer* to generate the interpolated view in a single feed-forward pass. Figure 2 shows an overview of the conventional video stitching pipeline and our proposed method. Our method yields results that are visually superior to existing solutions, both academic and commercial, as we show with an extensive user study.

2 Related Work

Image stitching. Existing image stitching methods often build on the conventional pipeline



(a) Conventional video stitching pipeline [12]



(b) Proposed pushbroom stitching network

Figure 2: **Algorithm overview.** (a) Conventional video stitching algorithms [12] use spatio-temporal local mesh warping and 3D graph cut to align the entire video, which are often sensitive to scene content and computationally expensive. (b) The proposed pushbroom stitching network adopts a pushbroom interpolation layer to gradually align the input views, and outperforms prior work and commercial software.

of Brown and Lowe [4], which first estimates a 2D transformation (*e.g.*, homography) for alignment and then stitches the images by defining seams [9] and using multi-band blending [5]. However, ghosting artifacts and mis-alignment still exist, especially when input images have large parallax. To account for parallax, several methods adopt spatially varying local warping based on the affine [18] or projective [30] transformations. Zhang *et al*. [31] integrate the content-preserving warping and seam-cutting algorithms to handle parallax while avoiding local distortions. More recent methods combine the homography and similarity transforms [6, 16] to reduce the projective distortion (*i.e.*, stretched shapes) or adopt a global similarity prior [7] to preserve the global shape of the resulting stitched images.

While these methods perform well on still images, applying them to videos frame-by-frame results in strong temporal instability. In contrast, our algorithm, also single-frame, generates videos that are spatio-temporally coherent, because our pushbroom layer only operates on the overlapping regions, while the rest is directly taken from the inputs.

Video stitching. Due to computational efficiency, it is not straightforward to enforce spatio-temporal consistency in existing image stitching algorithms. Commercial software, *e.g.*, VideoStitch Studio [27] or AutoPano Video [3], often finds a fixed transformation (with camera calibration) to align all the frames, but cannot align local content well. Recent methods integrate local warping and optical flow [21] or find a spatio-temporal content-preserving warping [12] to stitch videos, which are computationally expensive. Lin *et al*. [17] stitch videos captured from hand-held cameras based on 3D scene reconstruction, which is also time-consuming. On the other hand, several approaches, *e.g.*, Rich360 [15] and Google Jump [2], create 360° videos from multiple videos captured on a structured rig. Recently, NVIDIA released VRWorks [19], a toolkit to efficiently stitch videos based on depth and motion estimation. Still, as shown in Figure 1(b), several artifacts, *e.g.*, broken objects and ghosting, are visible in the stitched video.

In contrast to existing video stitching methods, our algorithm learns local warping flow fields based on a deep CNN to effectively and efficiently align the input views. The flow is learned to optimize the quality of the stitched video in an end-to-end fashion.

Pushbroom panorama. Linear pushbroom cameras [10] are common for satellite imagery:

while a satellite moves along its orbit, they capture multiple 1D images, which can be concatenated into a full image. A similar approach has also been used to capture street view images [24]. However, when the scene is not planar, or cannot be approximated as such, they introduce artifacts, such as stretched or compressed objects. Several methods handle this issue by estimating scene depth [22], finding a cutting-seam on the space-time volume [29], or optimizing the viewpoint for each pixel [1]. The proposed method is a software simulation of a pushbroom camera which creates panoramas by concatenating vertical slices that are spatially interpolated between the input views. We note that the method of Jin *et al*. [13] addresses a similar problem of view morphing, which aims at synthesizing intermediate views along a circular path. However, they focus on synthesizing a single object, *e.g*. a person or a car, and do not consider the background. Instead, our method synthesizes intermediate views for the entire scene.

3 Stitching as Spatial Interpolation

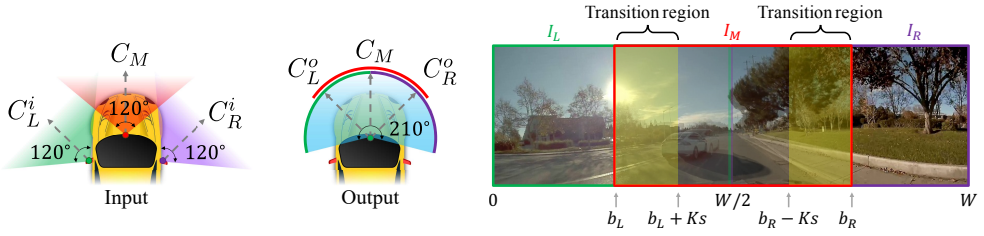
Our method produces a temporally stable stitched video from wide-baseline inputs of dynamic scenes. While the proposed approach is suitable for a generic linear camera array configuration, here we describe it with reference to the automotive use case. Unlike other applications of structured camera arrays, in the automotive case, objects can come arbitrarily close to the cameras, thus requiring the stitching algorithm to tolerate large parallax.

For the purpose of describing the method, we define the camera setup as shown in Figure 3(a), which consists of three fisheye cameras whose baseline spans the entire car's width. Figures 4(a)-(c) show typical images captured under this configuration, and underscore some of the challenges we face: strong parallax, large exposure differences, as well as geometric distortion. To minimize the appearance change between the three views and to represent the wide FOV of the stitched frames, we first adopt a camera pose transformation to warp C_L^i and C_R^i to the position of C_L^o and C_R^o, respectively. Therefore, the new origin is set at the center camera C_M. Then, we apply a cylindrical projection (by approximating the scene to be at infinity) to warp all the views onto a common viewing cylinder, as shown in Figure 3(a). However, even after camera calibration, exposure compensation, fisheye distortion correction, and cylindrical projection, parallax still causes significant misalignment, which results in severe ghosting artifacts, as shown in Figure 4(d).

3.1 Formulation

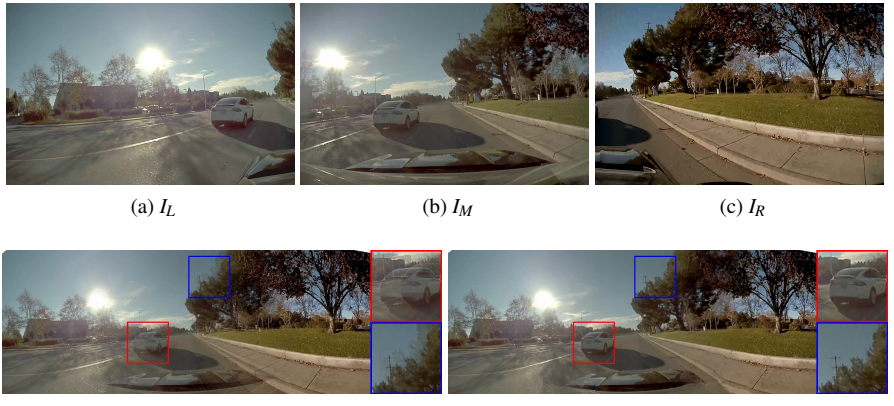
In this work, we cast video stitching as a problem of spatial interpolation between the side views and the center view. We denote the output view by $\mathcal{O}$, and the input views (projected onto the output cylinder) by I_L, I_M, and I_R, respectively. Note that I_L, I_M, and I_R are in the same coordinate system and have the same resolution. We define a *transition region* as part of the overlapping region between a pair of inputs (see the yellow regions in Figure 3(b)). Within the transition region, we progressively warp K vertical slices from both images to create a smooth transition from one camera to another. Outside the transition region, we directly take the pixel values from the input images without modifying them.

For presentation clarity, here we focus only on I_L and I_M. Our goal is to generate K intermediate frames, $\hat{I}_L^{(k)}$, which smoothly transition between I_L and I_M. We first compute the bidirectional optical flows, $F_{L\to M}$ and $F_{M\to L}$, and then generate warped frames $\hat{I}_L^{(k)} = \mathcal{W}(I_L, \alpha_k \cdot F_{L\to M})$ and $\hat{I}_M^{(k)} = \mathcal{W}(I_M, (1-\alpha_k) \cdot F_{M\to L})$, where $\mathcal{W}(I,F)$ is a function



(a) Camera setup (b) Images on the viewing cylinder

Figure 3: **Camera setup and input images.** (a) The input videos are captured from three fisheye cameras, C_L^i, C_M, and C_R^i, separated by a large baseline. The output is a viewing cylinder centered at C_M. (b) The input images are first projected on the output cylinder assuming a constant depth. Within the transition regions, our pushbroom interpolation method progressively warps and blends K vertical slices from the input views to create a smooth transition. Outside the transition regions, we do not modify the content from the inputs.



(a) I_L (b) I_M (c) I_R

(d) Images blended on the cylinder (e) Pushbroom interpolation

Figure 4: **Example of input and stitched views.** Simply projecting the input views I_L, I_M, and I_R onto the output cylinder causes artifacts due to the parallax and scene depth variation (d). Our pushbroom interpolation method effectively stitches the views and does not produce ghosting artifacts (e).

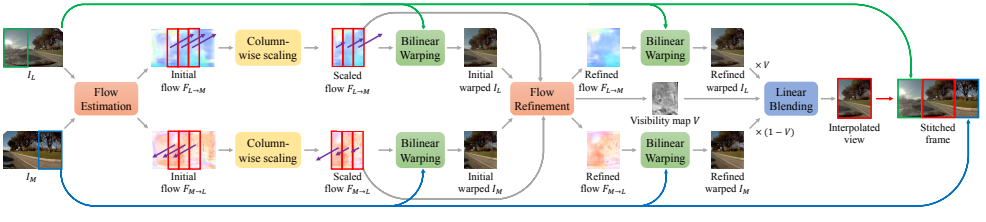


Figure 5: **Pushbroom interpolation layer.** A straightforward implementation of the pushbroom interpolation layer requires to generate all the intermediate flows and the intermediate views, which is time-consuming when the number of interpolated views K is large. Therefore, we develop a fast pushbroom interpolation layer by a column-wise scaling on optical flows, which only requires to generate one interpolated image for any given K.

that warps image I based on flow F, and $\alpha_k = \{k/K\}_{k=1,\ldots,K}$ scales the flow to create the smooth transition. We define the left stitching boundary b_L as the column of the leftmost valid pixel for I_M on the output cylinder. Given the interpolation step size s, the left half of the output view, $\mathcal{O}_L$, is constructed by

$$\mathcal{O}_L(x) = \begin{cases} I_L(x), & \text{if } 0 \le x < b_L \\ \hat{I}^{(k)}_{LM}(x), & \text{if } b_L + (k-1)\cdot s \le x < b_L + k\cdot s \\ I_M(x), & \text{if } b_L + K\cdot s \le x < \frac{W}{2}, \end{cases} \tag{1}$$

where W is the width of the output frame, and $\hat{I}^{(k)}_{LM}$ is obtained by appropriately fusing $\hat{I}^{(k)}_L$ and $\hat{I}^{(k)}_M$, $k = 1,\ldots,K$ (see Section 3.2). By construction, the output image is aligned with I_L at b_L, and aligned with I_M at $b_L + K\cdot s$. Within the transition region, the output view gradually changes from I_L to I_M by taking the corresponding columns from the intermediate views. The right half part of the output, $\mathcal{O}_R$, is defined similarly to $\mathcal{O}_L$. Figure 4(e) shows a result of this interpolation.

We note that the finer the interpolation steps, the higher the quality of the stitched results. We empirically set $K = 100$ and $s = 2$, *i.e.*, 100 pushbroom columns each 2-pixel wide.

3.2 Fast Pushbroom Interpolation Layer

Synthesizing the transition regions exactly as described in the previous section is computationally expensive. For each side, it requires scaling the forward and backward optical flow fields K times, and using them to warp the full-resolution images just as many times. For $H \times W$ images, this results in $2 \times H \times W \times K$ pixels to warp for each side. However, we only need a slice of $s = 2$ pixels from each of them.

Instead of scaling each flow field in its entirety, we propose to generate a single flow field in which entries corresponding to different slices are scaled differently. For instance, from the flow field from I_L to I_M, we generate a new field

$$\hat{F}_{L\to M}(x) = \begin{cases} 0, & \text{if } 0 \le x < b_L \\ \alpha_k F_{L\to M}(x), & \text{if } b^{(k)} \le x < b^{(k+1)} \\ F_{L\to M}(x), & \text{if } b_L + K\cdot s \le x < \frac{W}{2}, \end{cases} \tag{2}$$

where $b^{(k)} = b_L + (k-1)\cdot s$ are the boundaries of each slice. We can then warp both images as $\hat{I}_L = \mathcal{W}(I_L, \hat{F}_{L\to M})$ and $\hat{I}_M = \mathcal{W}(I_M, \hat{F}_{M\to L})$, where $\hat{F}_{M\to L}$ is computed with Equation 2 with $(1-\alpha_k)$ in place of α_k. Note that this approach only warps each pixel in the input images *once*.

To deal with the unavoidable artifacts of optical flow estimation, we use a flow refinement network to refine the scaled flows and predict a visibility map for blending. As shown in Figure 5, the flow refinement network takes the scaled optical flows and the initial estimates of the warped images, from which it generates refined flows and a visibility map V. The visibility map can be considered as a quality measure of the flow, which prevents any potential ghosting artifacts due to occlusions. With the refined flows, we warp the input images again to obtain $\tilde{I}_L$ and $\tilde{I}_M$. The final interpolated image is then generated by blending based on visibility: $\tilde{I}_{LM} = V \cdot \tilde{I}_L + (1-V)\cdot \tilde{I}_M$.

Finally, the output view, $\mathcal{O}_L$, is constructed by replacing all the $\hat{I}^{(k)}_{LM}$ in Equation 1 with $\tilde{I}_{LM}$. We generate $\tilde{I}_{RM}$ and construct $\mathcal{O}_R$ by mirroring the process above. Our fast pushbroom

interpolation layer generates the results with similar quality but is about 40× faster than the direct implementation for an output image with a resolution of 1000×600 pixels.

3.3 Training Pushbroom Stitching Network

Training dataset. Capturing data to train our system is challenging, as one would need to use hundreds of synchronized linear cameras. Instead, we render realistic synthetic data using the urban driving simulator CARLA [8], which allows us to specify the location and rotation of cameras. For the input cameras, we follow the setup of Figure 3(a). To synthesize the output pushbroom camera, we use 100 cameras uniformly spaced between C_L^i and C_M, and between C_R^i and C_M. We then use Equation 1 and replace $I_{LM}^{(k)}$ with these views to render ground-truth stitched video. We synthesize 152 such videos with different routes and weather conditions (*e.g.*, sunny, rainy, cloudy, etc.) for training. We provide the detailed network architectures of the flow estimation and flow refinement networks in the supplementary material.

Training loss. To train our pushbroom interpolation network, we optimize the following loss functions: (1) content loss, (2) perceptual loss, and (3) temporal warping loss.

The content loss is computed by $\mathcal{L}_C = \sum_{x,y} M_{x,y} \cdot \|\mathcal{O}_{x,y} - \mathcal{S}_{x,y}\|_1$, where $\mathcal{O}$ is the output image, $\mathcal{S}$ is the ground-truth, and $M_{x,y}$ is a mask indicating whether pixel x,y is valid on the viewing cylinder. The perceptual loss is computed by $\mathcal{L}_P = \sum_{x,y} \tilde{M}_{x,y} \cdot \|\phi_{x,y}(\mathcal{O}) - \phi_{x,y}(\mathcal{S})\|_1$, where $\phi(\cdot)$ denotes the feature activation at the `relu4-3` layer of the pre-trained VGG-19 network [25] and $\tilde{M}$ is the valid mask downscaled to the size of the corresponding features. To improve the temporal stability, we also optimize the temporal warping loss [14] $\mathcal{L}_T = \sum_{t' \in \Omega_t} \sum_{x,y} M_{x,y} \cdot C_{x,y}^{t \Rightarrow t'} \cdot \|\mathcal{O}_{x,y}^{(t)} - \hat{\mathcal{O}}_{x,y}^{(t')}\|_1$, where Ω_t is the set of neighboring frames at time t, C is a confidence map, and $\hat{\mathcal{O}}^{(t')} = \mathcal{W}(\mathcal{O}^{(t')}, F^{t \Rightarrow t'})$ is the frame warped with optical flow $F^{t \Rightarrow t'}$. We use PWC-Net [26] to compute the optical flow between subsequent frames. Note that the optical flow $F^{t \Rightarrow t'}$ is only used to compute the training loss, and is not needed at testing time. The confidence map $C^{t \Rightarrow t'} = \exp(-\alpha \|\mathcal{S}^{(t)} - \hat{\mathcal{S}}^{(t')}\|_2^2)$ is computed from the ground-truth frame $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(t')}$, where $C \in [0,1]$. A smaller value of C indicates that the pixel is more likely to be occluded.

The overall loss function is defined as $\mathcal{L} = \lambda_C \mathcal{L}_C + \lambda_P \mathcal{L}_P + \lambda_T \mathcal{L}_T$, where λ_C, λ_P, and λ_T are balancing weights set empirically. We empirically set $\lambda_C = 1$, $\lambda_P = 0.001$, and $\lambda_T = 10$. For the spatial optical flow in the transition regions, we use SuperSloMo [11] initialized with the weights provided by the authors and then fine-tuned to our use-case in our end-to-end training. We provide more implementation details in the supplementary material.

4 Experimental Results

The output of our algorithm, while visually pleasing, does not match a physical optical system, since the effective projection matrix changes horizontally. To numerically evaluate our results we can use rendered data (Section 4.1). However, a pixel-level numerical comparison with other methods is not possible as each method effectively uses a different projection matrix. For a fair comparison, then, we carried out an extensive user study (Section 4.2). The video results are available in the supplementary material and our project website at http://vllab.ucmerced.edu/wlai24/video_stitching/.

(a) Stitched frame ($\mathcal{L}_c$+$\mathcal{L}_p$+$\mathcal{L}_t$) (b) GT (c) Baseline (d) $\mathcal{L}_c$ (e) $\mathcal{L}_c$+$\mathcal{L}_p$ (f) $\mathcal{L}_c$+$\mathcal{L}_p$+$\mathcal{L}_t$

Figure 6: **Stitching on a synthetic video.** After training the proposed model on the synthetic data, our model aligns the content well and reduces ghosting artifacts.

Training loss	PSNR ↑	SSIM ↑	E_{warp} ↓
N.A. (baseline)	27.69	0.908	13.89×10^{-4}
$\mathcal{L}_C$	30.72	0.925	11.72×10^{-4}
$\mathcal{L}_C + \mathcal{L}_P$	31.04	0.926	11.63×10^{-4}
$\mathcal{L}_C + \mathcal{L}_P + \mathcal{L}_T$	31.27	0.928	10.67×10^{-4}

Table 1: **Ablation study.** After training the model with the content loss $\mathcal{L}_C$, perceptual loss $\mathcal{L}_P$, and the temporal loss $\mathcal{L}_T$, the image quality and temporal stability are significantly improved.

Ours vs.	Preference	Broken objects	Less ghosting	Similar results
AutoPano [3]	90.74%	85.71%	20.41%	10.20%
VRWorks [19]	97.22%	80.00%	49.52%	1.90%
STCPW [12]	98.15%	87.74%	38.68%	0%
Overall	95.37%	84.74%	36.57%	3.88%

Table 2: **User study.** We conduct pairwise comparisons on 20 real videos. Our method is preferred by 95% of users on average.

4.1 Model Analysis

To quantitatively evaluate the performance of the stitching quality, we use the CARLA simulator to render a test set using a different town map from the training data. We render 10 test videos, where each video has 300 frames.

We measure the PSNR and SSIM [28] between the stitched frames and the ground-truth images for evaluating the image quality. In addition, we measure the temporal stability by computing the temporal warping error $E_{\text{warp}} = \sum_{t=1}^{T-1} \frac{1}{|\tilde{M}^{(t)}|} \sum_{x,y \in \tilde{M}^{(t)}} \| O_{x,y}^{(t)} - \hat{O}_{x,y}^{(t+1)} \|_2^2$, where $\hat{O}^{(t+1)}$ is the flow-warped frame $O^{(t+1)}$, $\tilde{M}^{(t)}$ is a mask indicating the non-occluded pixels, and $|\tilde{M}^{(t)}|$ is the number of valid pixels in the mask. We use the occlusion detection method by Ruder *et al*. [23] to estimate the mask $\tilde{M}^{(t)}$.

We first evaluate the baseline model, where the pushbroom interpolation layer is initialized with the pre-trained SuperSloMo [11]. The baseline model provides a visually plausible stitching result but causes object distortion and temporal flickering due to inaccurate flow estimation. After fine-tuning the whole model, both the visual quality and temporal stability are significantly improved. As shown in Table 1, all the loss functions, $\mathcal{L}_C$, $\mathcal{L}_P$, and $\mathcal{L}_T$, improve the PSNR and SSIM and also reduce the temporal warping error. In Figure 6, we show an example where our full model aligns the speed sign well and avoids the ghosting artifacts.

Figure 7(a) shows a stitched frame from the baseline model, where the pole on the right is distorted and almost disappears. After training, the pole remains intact, Figure 7(b). We also visualize the optical flows before and after training the model. After end-to-end training, the flows are smoother and warp the pole as a whole, avoiding distortion.

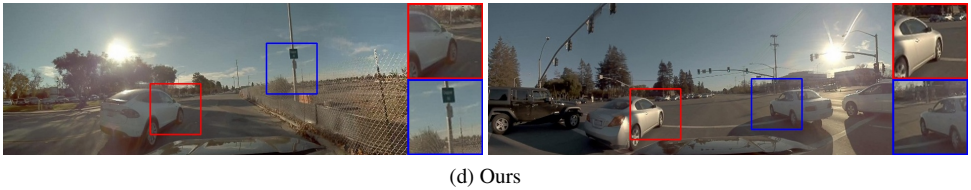
(a) (b) (c)

Figure 7: **Visualization of the stitched frames and flows.** We show the stitched frames (a), forward flows (b), and backward flows (c) from the pushbroom interpolation layer before (top) and after (bottom) fine-tuning the proposed model. The fine-tuned model generates smooth flow fields to warp the input views and preserve the content (e.g., the pole on the right) well.

(a) STCPW [12]

(b) AutoPano Video [3]

(c) NVIDIA VRWorks [19]



(d) Ours

Figure 8: **Comparison with existing video stitching methods.** The proposed method achieves better alignment quality while better preserving the shape of objects and avoiding ghosting artifacts.

4.2 Comparisons with Existing Methods

We compare the proposed method with commercial software, AutoPanoVideo [3], and existing video stitching algorithms, STCPW [12] and NVIDIA VRWorks [19]. We show two stitched frames from real videos in Figure 8, where the proposed approach generally achieves better alignment quality with fewer broken objects and ghosting artifacts. More video results are provided in the supplementary material.

As different methods use different projection models, a fair quantitative evaluation of the different video stitching algorithms is impossible. Therefore, we conduct a human subject study through pairwise comparisons. Specifically, we ask the participants to indicate which stitched video presents fewer artifacts from a pair of videos. We evaluate a total of 20 real videos and ask each participant to compare 12 pairs of videos. In each comparison, participants can watch both videos for multiple times before making a selection. In total, we collect the results from 54 participants.

Table 2 shows that our results are preferred by about 95% of users, which demonstrates the effectiveness of the proposed method on generating high-quality stitching results. In addition, we ask participants to provide the reasons why they prefer the selected video from the following options: (1) the video has fewer broken lines or objects, (2) the video has less ghosting artifacts, and (3) the two videos are similar. Overall, our results are preferred due to better alignment and fewer broken objects. Moreover, only 4% of users feel that our result is comparable to the others, which indicates that users generally have a clear judgment when comparing our method with other approaches.

4.3 Discussion and Limitations

Our method requires the cameras to be calibrated for the cylindrical projection of the inputs. While common to many stitching methods, *e.g.*, NVIDIA's VRWorks [19], this requirement can be limiting, if strict. However, our experiments reveal that moving the side cameras inwards by up to 62.5% of the original baseline, reduces the PSNR by less than 1dB. An outward shift is more problematic because it reduces the overlap between the views. Still, an outward shift that is 30% of the original baseline causes less than 2dB drop in PSNR. Fine-tuning the network by perturbing the original configuration of cameras can reduce the error. We present detailed analysis in the supplementary material.

Our method inherits some limitations of the optical flow. For instance, thin structures can cause a small amount of ghosting effects. We show failure cases in the supplementary material. In practice, the proposed method performs robustly even in such cases.

5 Conclusion

In this work, we present an efficient algorithm to stitch videos with deep CNNs. We propose to cast video stitching as a problem of spatial interpolation and we design a pushbroom interpolation layer for this purpose. Our model effectively aligns and stitches different views while preserving the shape of objects and avoiding ghosting artifacts. To the best of our knowledge, ours is the first learning-based video stitching algorithm. A human subject study demonstrates that it outperforms existing algorithms and commercial software.

References

[1] Aseem Agarwala, Maneesh Agrawala, Michael Cohen, David Salesin, and Richard Szeliski. Photographing long scenes with multi-viewpoint panoramas. *ACM TOG*, 2006. 4

[2] Robert Anderson, David Gallup, Jonathan T Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. Jump: virtual reality video. *ACM TOG*, 2016. 3

[3] AutoPano Video. http://www.kolor.com/. 1, 2, 3, 8, 9, 10

[4] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *IJCV*, 2007. 3

[5] Peter J Burt and Edward H Adelson. A multiresolution spline with application to image mosaics. *ACM TOG*, 1983. 2, 3

[6] Che-Han Chang, Yoichi Sato, and Yung-Yu Chuang. Shape-preserving half-projective warps for image stitching. In *CVPR*, 2014. 2, 3

[7] Yu-Sheng Chen and Yung-Yu Chuang. Natural image stitching with the global similarity prior. In *ECCV*, 2016. 3

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning*, 2017. 7

[9] Ashley Eden, Matthew Uyttendaele, and Richard Szeliski. Seamless image stitching of scenes with large motions and exposure differences. In *CVPR*, 2006. 2, 3

[10] Rajiv Gupta and Richard I Hartley. Linear pushbroom cameras. *TPAMI*, 1997. 2, 3

[11] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In *CVPR*, 2018. 7, 8

[12] Wei Jiang and Jinwei Gu. Video stitching with spatial-temporal content-preserving warping. In *CVPR Workshops*, 2015. 1, 2, 3, 8, 9, 10

[13] Shi Jin, Ruiynag Liu, Yu Ji, Jinwei Ye, and Jingyi Yu. Learning to dodge a bullet: Concyclic view morphing via deep learning. In *ECCV*, 2018. 4

[14] Wei-Sheng Lai, Jia-Bin Huang, Oliver Wang, Eli Shechtman, Ersin Yumer, and Ming-Hsuan Yang. Learning blind video temporal consistency. In *ECCV*, 2018. 7

[15] Jungjin Lee, Bumki Kim, Kyehyun Kim, Younghui Kim, and Junyong Noh. Rich360: Optimized spherical representation from structured panoramic camera arrays. *ACM TOG*, 2016. 3

[16] Chung-Ching Lin, Sharathchandra U Pankanti, Karthikeyan Natesan Ramamurthy, and Aleksandr Y Aravkin. Adaptive as-natural-as-possible image stitching. In *CVPR*, 2015. 3

[17] Kaimo Lin, Shuaicheng Liu, Loong-Fah Cheong, and Bing Zeng. Seamless video stitching from hand-held camera inputs. In *EG*, 2016. 3

[18] Wen-Yan Lin, Siying Liu, Yasuyuki Matsushita, Tian-Tsong Ng, and Loong-Fah Cheong. Smoothly varying affine stitching. In *CVPR*, 2011. 3

[19] Nvidia VRWorks. https://developer.nvidia.com/vrworks/vrworks-360video. 1, 2, 3, 8, 9, 10

[20] Shmuel Peleg and Joshua Herman. Panoramic mosaics with videobrush. In *CVPR*, 1997.

[21] Federico Perazzi, Alexander Sorkine-Hornung, Henning Zimmer, Peter Kaufmann, Oliver Wang, S Watson, and M Gross. Panoramic video from unstructured camera arrays. In *Computer Graphics Forum*, 2015. 2, 3

[22] Alex Rav-Acha, Yael Shor, and Shmuel Peleg. Mosaicing with parallax using time warping. In *CVPR Workshops*, 2004. 4

[23] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *German Conference on Pattern Recognition*, 2016. 8

[24] Steven M Seitz and Jiwon Kim. Multiperspective imaging. *IEEE Computer Graphics and Applications*, 2003. 4

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 7

[26] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 7

[27] VideoStitch Studio. https://www.orah.co/news/videostitch-studio. 2, 3

[28] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *TIP*, 2004. 8

[29] Yonatan Wexler and Denis Simakov. Space-time scene manifolds. In *ICCV*, 2005. 4

[30] Julio Zaragoza, Tat-Jun Chin, Michael S Brown, and David Suter. As-projective-as-possible image stitching with moving dlt. In *CVPR*, 2013. 2, 3

[31] Fan Zhang and Feng Liu. Parallax-tolerant image stitching. In *CVPR*, 2014. 2, 3