MeshFlow: Minimum Latency Online Video Stabilization

Shuaicheng Liu[1(✉)], Ping Tan[2], Lu Yuan[3], Jian Sun[3], and Bing Zeng[1]

[1] University of Electronic Science and Technology of China, Chengdu, China
{liushuaicheng,eezeng}@uestc.edu.cn
[2] Simon Fraser University, Burnaby, Canada
pingtan@sfu.ca
[3] Microsoft Research Asia, Beijing, China
{luyuan,jiansun}@microsoft.com

Abstract. Many existing video stabilization methods often stabilize videos off-line, i.e. as a postprocessing tool of pre-recorded videos. Some methods can stabilize videos online, but either require additional hardware sensors (e.g., gyroscope) or adopt a single parametric motion model (e.g., affine, homography) which is problematic to represent spatially-variant motions. In this paper, we propose a technique for online video stabilization with only *one* frame latency using a novel *MeshFlow* motion model. The MeshFlow is a spatial smooth sparse motion field with motion vectors only at the mesh vertexes. In particular, the motion vectors on the matched feature points are transferred to their corresponding nearby mesh vertexes. The MeshFlow is produced by assigning each vertex an unique motion vector via two median filters. The path smoothing is conducted on the *vertex profiles*, which are motion vectors collected at the same vertex location in the MeshFlow over time. The profiles are smoothed adaptively by a novel smoothing technique, namely the *Predicted Adaptive Path Smoothing* (PAPS), which only uses motions from the past. In this way, the proposed method not only handles spatially-variant motions but also works online in real time, offering potential for a variety of intelligent applications (e.g., security systems, robotics, UAVs). The quantitative and qualitative evaluations show that our method can produce comparable results with the state-of-the-art off-line methods.

Keywords: Online video stabilization · MeshFlow · Vertex profile

1 Introduction

Most existing video stabilization methods stabilize videos offline [1–5], where the videos have already been recorded. These methods post-process shaky videos by estimating and smoothing camera motions for the stabilized results. Typically, to stabilize the motion at each time instance, they require not only camera motions in the past but also camera motions in the future for high quality stabilization. There is an increasing demand of online video stabilization, where the video

B. Leibe et al. (Eds.): ECCV 2016, Part VI, LNCS 9910, pp. 800–815, 2016.
DOI: 10.1007/978-3-319-46466-4_48

is stabilized on the spot during capturing. For example, a robot or drone often carries a wireless video camera so that a remote operator is aware of the situation. Ideally, the operator wants to see the video stabilized as soon as it appears on the monitor for immediate responses. Offline stabilization are not suitable for this application, though they produce strongly stabilized results.

Online stabilization is challenging mainly for two reasons. Firstly, the camera motion estimation is difficult. Some online stabilization methods use gyroscope [6,7] for realtime motion estimation. However, gyro-based methods can only capture rotational motion, leaving translational motion untouched. High quality video stabilization requires handling of spatially-variant motion, which is often due to parallax and camera translation, a common problem in general scenes with depth changes. Spatially-variant motion is complicated. It cannot be represented by a single homography [1,3]. Recent methods [4,5,8] divide the video frame into several regions. However, this strategy is computationally expensive and hinders realtime applications. Enforcing spatial-temporal coherence during camera motion smoothing further complicates this approach.

Secondly, successful camera motion filtering often requires future frames. Some online video stabilization methods [9–11] use the single homography model and buffer some future frames. For example, the method of [10] requires a minimum of one second delay. The temporal buffer is needed to adaptively set the smoothing strength so as to avoid artifacts caused by excessive smoothing. Reducing this buffer for future frame will significantly deteriorate the results.

We design an online video stabilization method with minimum latency by solving the two aforementioned challenges. Our method only requires past motions for high quality motion filtering. We propose a novel motion model, *MeshFlow*, which is a spatially smooth sparse motion field with motion vectors defined only at the mesh vertexes. It can be regarded as a down-sampled dense flow. Specifically, we place a regular 2D mesh on the video frame. We then track image corners between consecutive frames, which yields a motion vector at each feature location. Next, these motion vectors are transferred to their corresponding nearby mesh vertexes, such that each vertex accumulates several motions from its surrounding features. The MeshFlow is a sparse 2D array of motion vectors consisting of motions at all mesh vertices.

With regards to the camera motion smoothing, we design a filter to smooth the temporal changes of the motion vector at each mesh vertex. This filter is applied to each mesh vertex. Thus, it can naturally deal with the spatially-variant motion. The uniqueness of this filter is that it mainly requires previous motions for strong stabilization. This is achieved by predicting an appropriate smoothing strength according to the camera motion at previous frames. In this way, it can achieve adaptive smoothing to avoid excessive cropping and wobble distortions. We call this filter *Predicted Adaptive Path Smoothing* (PAPS).

In summary, the main contribution of the paper consists of: (1) a computationally efficient motion model, MeshFlow, for spatially-variant motion representation; and (2) an adaptive smoothing method PAPS, designed for the new model for online processing with only one frame latency. We evaluate our method

on various challenging videos and demonstrate its effectiveness in terms of both visual quality and efficiency.[1]

2 Related Work

According to the adopted motion model, video stabilization methods can be categorised into 3D [8,12,13], 2D [1,3,14], and 2.5D [2,15] approaches.

The 3D methods estimate camera motions in 3D space for stabilization. Beuhler *et al.* [16] stabilized videos under projective 3D reconstruction. Liu *et al.* [8] applied Structure from Motion (SfM) to the video frames and used content preserving warps for novel view synthesis. Zhou *et al.* [13] introduced 3D plane constraints for improved warping quality. Smith *et al.* [17] and Liu *et al.* [5] adopted light field camera and Kinect camera, respectively, in acquiring of 3D structures. Methods [6,7] and [18] used gyroscope to estimate 3D rotations. Some 2.5D approaches relax the full 3D requirement to some partial 3D information that is embedded in long feature tracks. Goldstein and Fattal [15] used "epipolar transfer" to enhance the length of feature tracks while Liu *et al.* [2] smoothed feature tracks in subspace so as to maintain the 3D constraints. Later, the subspace approach is extended for stereoscopic videos [19]. All these methods either conducted expensive and brittle 3D reconstruction or required additional hardware sensors for stabilization. In contrast, our method is a sensor-free approach that neither recoveries the 3D structures nor relies on long feature tracks.

The 2D methods use a series of 2D linear transformations (e.g., affines, homographies) for motion estimation and smooth them for stabilized videos [1,20–22]. Grundmann *et al.* [3] employed cinematography rules for camera path design. Later, they extended their approach by dividing a single homography into homography array [14] such that the rolling shutter distortions could be well compensated. Wang *et al.* [4] divided frames into triangles and smoothed feature trajectories with a spatial-temporal optimization. Liu *et al.* [5] smoothed bundled paths for spatially-variant motions. Bai *et al.* [23] extended the bundled-paths by introducing user interactions. Liu *et al.* [24] proposed to replace the smoothing of feature tracks with the smoothing of pixel profiles and showed several advantages over smoothing of traditional feature tracks. Inspired from [24], we propose to smooth vertex profiles, a sparse version of pixel profiles, for an improved robustness and efficiency, which facilitates an online system with spatially-variant motion representation.

3 MeshFlow

In this section, we introduce the MeshFlow motion model. Figure 1 shows a comparison between the SteadyFlow [24] and our MeshFlow. Compared with the SteadyFlow, which calculates dense optical flow and extracts *pixel profiles* at all pixel locations for stabilization, our MeshFlow is computationally more efficient.

[1] Project page: http://www.liushuaicheng.org/eccv2016/index.html.

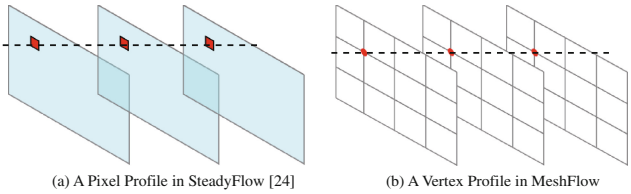

Fig. 1. (a) Pixel profiles [24] collect motion vectors at the same pixel location in SteadyFlow over time for all pixel locations. Motions of SteadyFlow come from dense optical flow. (b) Vertex profiles only collect motion vectors in MeshFlow at mesh vertexes. Motions of MeshFlow come from feature matches between adjacent frames.

We only operate on a sparse regular grid of *vertex profiles*, such that the expensive optical flow can be replaced with cheap feature matches. For one thing, they are similar because they both encode strong spatial smoothness. For another, they are different as one is dense and the other is sparse. Moreover, the motion estimation methods are totally different. Next, we show how to estimate spacial coherent motions at mesh vertexes.

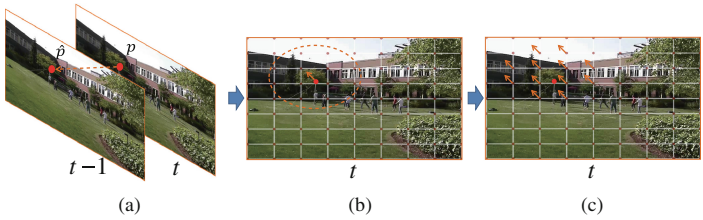

Fig. 2. (a) A pair of matched features (red dots) between frame t and $t-1$. (b) The arrow indicates the motion of the feature point at frame t. (c) The motion is propagated to the nearby vertexes. (Color figure online)

3.1 Motion Propagation

We match image features between neighboring frames. Figure 2 shows an example. Suppose $\{p, \hat{p}\}$ is the p-th matched feature pair, with p at frame t and $\hat{p}$ at frame $t-1$ (p and $\hat{p}$ denote the image coordinates of features). The motion v_p at feature location p can be computed as: $v_p = p - \hat{p}$ (see the dashed arrow in Fig. 2(a)). The mesh vertexes nearby the feature p should have a similar motion as v_p. Therefore, we define an eclipse that is centered at p (dashed circle in Fig. 2(b)) and assign v_p to the vertexes within the eclipse (see Fig. 2(c)). Specifically, we detect FAST features [25] and track them by KLT [26] to the adjacent frame. We place a uniform grid mesh with 16×16 regular cells onto each frame.[2]

[2] We draw this mesh as 8×8 in all figures for the purpose of clearer illustration.

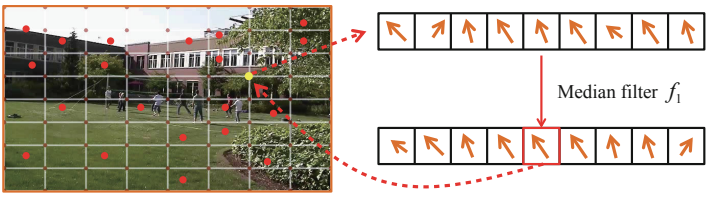

Fig. 3. A grid accumulates multiple motion vectors from several nearby features. We assign each grid a unique motion vector by applying median filter f_1 to the candidates. (Color figure online)

The eclipse covers 3×3 cells. Notably, according to literatures (such as works on visual SLAM [27,28]) and our own experiments, FAST and KLT are known as the most efficient way compared with other options (e.g., SURF, ORB etc.).

3.2 Median Filters

All matched features propagate their motions to their nearby mesh vertexes. Therefore, a vertex may receive multiple motion vectors. Figure 3 illustrates an example, where red dots denote feature points and a vertex (yellow dot) receives several motions. We propose to use a median filter f_1 to filter the candidates. The filter response is assigned to the vertex. The median filter is frequently used in optical flow estimation and has been treated as the secret of a high quality flow estimation [29]. Here, we borrow the similar idea for sparse motion regularization.

After applying f_1 to all vertexes, we obtain a sparse motion field as illustrated in the left part of Fig. 4. This motion field resembles the SteadyFlow, except that it is only defined at sparse mesh vertices. This sparsity is the key to make the spatially-variant motion estimation computationally lightweight. The motion field needs to be smooth spatially for stabilization [24]. However, due to various reasons, such as false feature matches and dynamic objects, the motion field is noisy (yellow arrows in the left part of Fig. 4). We propose to use another median filter f_2 (covers 3×3 cells) to remove the noises, producing a spatially-smooth sparse motion filed, MeshFlow, as shown in the right part of Fig. 4.

The two median filters provide the essential spatial smoothness to MeshFlow. The strong spatial smoothness is particularly important in flow-based stabilization [24]. Motion compensation on "noise flows" (inconsistent flows) often causes render artifacts on discontinuous boundaries [24]. Therefore, the SteadyFlow estimates dense raw optical flow [30] and upgrades it to a dense smooth flow by regularizing small vibrations and excluding large inconsistencies (i.e., flow on dynamic objects). Here, the two median filters achieve a similar effect: the first filter emphasizes on removing small noises and the second one concentrates on rejecting large outliers. More discussions are provided in Sect. 5.

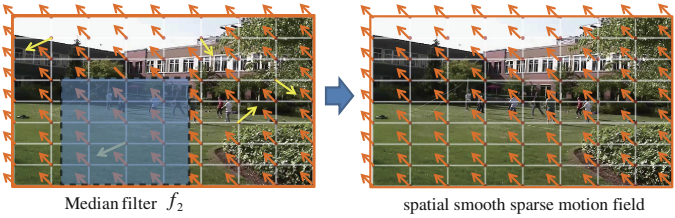

Fig. 4. A second median filter f_2 is applied to enforce spatial smoothness (Color figure online)

3.3 Vertex Profiles Generation

A vertex profile is generated by collecting all motion vectors at a vertex position in MeshFlow sequentially over time. It is a one-dimensional array consisting of motion vectors of all frames. Repeating the same procedure for all vertexes provides vertex profiles.

3.4 Robust Estimation

The robustness of MeshFlow estimation can be improved by several operations.

Rich features. We hope to have rich features to cover the entire video frame densely and uniformly such that every grid can receive several motion vectors. When selecting good feature points for tracking [26], a global threshold on corner response often produces few features in poorly textured regions (e.g., ground, sky), because the threshold is biased by other highly textured areas [14]. Therefore, we divide the image into small regions and adopt a local threshold.

Outlier rejection. Classical methods for outlier rejection, such as fitting a global homography model by RANSAC [31], is not applicable in our application, because we want to retain motions which do not reside in a global linear space. We divide the image into 4×4 sub-images and reject outliers with a local homography fitting by RANSAC. Large motion deviations caused by false matches or dynamic objects can be rejected successfully while small variations due to depth changes and rolling shutter effects [32,33] can be well maintained. A similar approach is reported in [5]. Notably, both median filters and the RANSAC are critical for the robust outlier removal. The former works locally while the latter reject outliers in a more global fashion.

Pre-warping. Before MeshFlow estimation, we use a global homography F_t, estimated using all matched features, to transform features $\hat{p}$ from frame $t-1$ to frame t. Clearly, F_t induces a global motion vector field V_t for all vertexes. The local residual motions are calculated as $\tilde{v}_p = p - F_t\hat{p}$. The MeshFlow is first estimated from these residual motions. Then, the final MeshFlow motion is an addition of the global and the local motions: $v_p = V_t + \tilde{v}_p$. A similar idea

is adopted in [5] for motion estimation and in [8] for view synthesis. For some extreme cases where no features are detected, (e.g., a purely white wall), vertexes are assigned with the global motion V_t.

4 Predicted Adaptive Path Smoothing (PAPS)

A vertex profile represents the motion of its neighboring image regions. We can smooth all the vertex profiles for the smoothed motions. We begin by describing an offline filter, and then extend it for online smoothing.

4.1 Offline Adaptive Smoothing

We can consider a vertex profile as the local camera path C_i. All vertex profiles aggregate to multiple camera paths $\mathbf{C}$ that covers the whole frame. We want to obtain the optimized paths $\mathbf{P}$. As the MeshFlow itself enforces strong spatial coherence, we do not enforce any additional spatial constraints during smoothing. That is, each vertex profile is smoothed independently, like the SteadyFlow [24].

For an aesthetic path optimization, we want to avoid excessive cropping as well as annoying distortions after stabilization. This can be achieved by leveraging the temporal smoothness and the similarity of the original paths:

$$\mathcal{O}(\mathbf{P}(t)) = \sum_t (\|\mathbf{P}(t) - \mathbf{C}(t)\|^2 + \lambda_t \sum_{r \in \Omega_t} w_{t,r} \|\mathbf{P}(t) - \mathbf{P}(r)\|^2) \tag{1}$$

where $\mathbf{C}(t) = \sum_t \mathbf{v}(t)$ is the camera path at time t $\big(\mathbf{v}(t)$ represents the MeshFlow at time t, $\mathbf{v}(0) = \mathbf{0}\big)$. The first term encourages the stabilized video staying close to the original camera path so as to avoid excessive cropping and distortions. The second term enforces temporal smoothness. Ω_t denotes a temporal smoothing radius, $w_{t,r}$ is a Gaussian weight which is set to $exp(-\|r-t\|^2/(\Omega_t/3)^2)$, and λ_t balances two terms. The energy function is quadratic and can be minimized by the sparse linear solver. Similar to approaches [5,24], we solve it iteratively by a Jacobi-based solver.

The adaptive weight λ_t for each frame is the most important component in Eq. 1. Adaptive controlling the strength of smoothness can effectively suppress some artifacts (e.g., large cropping, wobbling). It is a tradeoff between stability and some side-effects. If all λ_t is set to 0, the optimized path is equal to the original path such that the output video is identical to the input video with no cropping and wobbling. In general, smaller λ_t leads to less copping and wobbling.

The method in [5] adopted an iterative refinement approach to search the optimal value of λ_t for each frame. They proposed to evaluate the cropping and wobbling numerically. Suppose that we have the values of cropping and wobbling respect to all frames after the optimization by setting all $\lambda_t = 1$. Then, we check these values at each frame to see if they satisfy some pre-requirements. For example, at least 80 % of visual content must be maintained after cropping. For

any frame that does not satisfy the requirements, λ_t is decreased by a step and the optimization is re-run for a second time. The procedure is iterated until all frames satisfying the requirements. This dynamic parameter adjustment can find the optimal values of λ_t but is not efficient obviously. Notably, $w_{t,r}$ in [5] is a bilateral weight which leads to a quicker convergence than a Gaussian weight.

Though the above method is proven to be effective, it is designed for offline applications. It requires both previous and future frames. The iterative refinement for λ_t is impractical for the online scenario. Therefore, we propose to predict a reasonable value of λ_t rather than applying iterative adjustment.

4.2 Predict λ_t

We want to use the current camera motion to predict a suitable λ_t. Therefore, we need to find some indicators. Designing good indicators is non-trivial. By extensively experiments, we suggest two empirically good indicators, *translational element* T_v and *affine component* F_a, both extracted from a global homography F between adjacent frames.

Translational element measures the velocity of the current frame. It is calculated as $T_v = \sqrt{(v_x^2 + v_y^2)}$, where v_x and v_y represent motions in x and y directions in F, respectively. If we oversmooth frames under high velocity, the motion compensation often leads to large empty regions (black borders), resulting in excessive cropping. In iterative adjustment of λ_t, the cropping can only be evaluated afterwards. The translational element allows the cropping being evaluated beforehand.

Affine Component is computed by the ratio of the two largest eigenvalues of the affine part of the homography. A single homography can not describe the spatially-variant motion. For scenes with large depth variations, an estimated homography is highly distorted [34]. The distortion can be measured by its affine part. A similar idea is reported in [5] for distortion evaluation.

To reveal the relationship between the two proposed indicators and λ_t, we collect 200 videos with various camera motions and scene types from publicly available data sets [2,3,5,8,14,15]. We run the iterative adjustment algorithm for λ_t on these videos and record the corresponding values. In particular, the two indicators might be correlated. To better sketch their independent impact to λ_t, we use videos with quick camera motions to train the translational element while videos contain large depth variations are adopted for affine component.

The result is plotted in Fig. 5. Note that F_t is normalized by the image width and height. By observing the distributions, we fit two linear models:

$$\lambda_t' = -1.93 * T_v + 0.95 \tag{2}$$

$$\lambda_t'' = 5.83 * F_a + 4.88 \tag{3}$$

The final value of λ_t is chosen as $max(min(\lambda_t', \lambda_t''), 0)$. A lower λ_t can satisfy both requirements. Notably, the translational element and affine component are

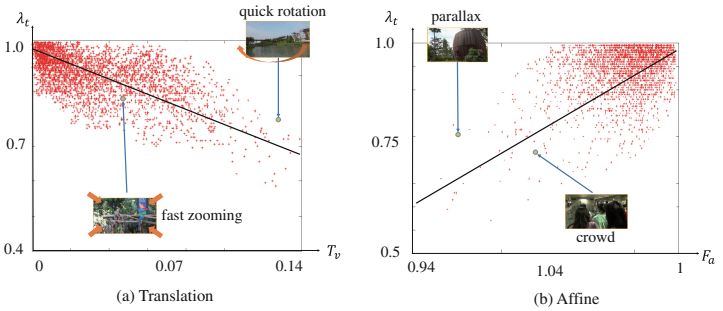

Fig. 5. (a) The plot of λ_t and values of translational elements T_v using 50 videos with quick camera motions (e.g., quick rotation, fast zooming). (b) The plot of λ_t and values of affine component F_a using another 50 videos containing large depth variations.

two indicators that we found empirically, which works well in practice. There might be some other alternatives. For example, more sophisticated method could be attempted for distortion evaluation, such as applying SfM to explicitly evaluate depth variations. Here, we keep our method simple and effective. Given λ_t for every frame, we run Eq. 1 to smooth vertex profiles. The optimization of Eq. 1 is efficient and can be further accelerated by the parallel computing.

4.3 Online Smoothing

The aforementioned approach is offline, though it can run in real time. Note that, processing an already captured video in a real time speed (e.g., 100 fps) is not identical to online stabilization. The online processing not only requires the real time speed, but also constrains the usage of future frames. Figure 6 shows our system setting. The green rectangle shows the current frame being displayed to the audiences. The red rectangle shows the incoming frame, which is about to show (turn into green) in the next run. The white rectangles are the past frames.

We define a buffer to hold previous frames. At the beginning, the buffer size is small, it increases gradually to a fixed size frame by frame and becomes a moving window that holds the latest frames and drops the oldest ones. *Each time*, we smooth the motions in the buffer by minimizing over the following energy:

$$\mathcal{O}(\mathbf{P}^{(\xi)}(t)) = \sum_{t\in\Phi}\left(\left\|\mathbf{P}^{(\xi)}(t)-\mathbf{C}(t)\right\|^2+\lambda_t\sum_{t\in\Phi,r\in\Omega_t}w_{t,r}\left\|\mathbf{P}^{(\xi)}(t)-\mathbf{P}^{(\xi)}(r)\right\|^2\right)$$
$$+\beta\sum_{t\in\Phi}\left\|\mathbf{P}^{(\xi)}(t-1)-\mathbf{P}^{(\xi-1)}(t-1)\right\|^2 \tag{4}$$

where Φ denotes the buffer. When an incoming frame arrives, we run the optimization of Eq. 4 (except the first two frames). The ξ indexes the optimization at each time. For each ξ, we obtain $\mathbf{P}^{(\xi)}(t)$ for all frames in the buffer. Only

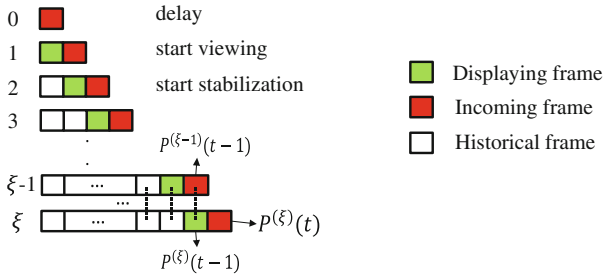

Fig. 6. The system is delayed at the time 0. At the time 1, we begin to display. The stabilization is started at the time 3 when there is at least one historical frame. The ξ indexes the optimization conducted at each time. The third term in Eq. 4 enforces similarities between paths in the current optimization ξ and paths obtained in the previous optimization $\xi-1$, which is indicated by the dashed lines. (Color figure online)

the result of the last one will be used to warp the incoming frame. All frames prior to that have already been stabilized and displayed. We can not change the paths of the displayed frames. We can only change the path of the current frame. Therefore, we add a third term to encourage the paths obtained at the current optimization ξ, up to $\mathbf{P}^{(\xi)}(t-1)$, to be similar to their previous optimized solutions obtained at the optimization $\xi-1$, i.e. $\mathbf{P}^{(\xi-1)}(t-1)$. The dashed line in Fig. 6 shows the paths enforced with similarity constraints. The β is a balancing weight which is set to 1.

Notably, Eq. 4 does not need any future motions (not even one) for optimization. However, our system has a one frame latency. Because no matter how fast the optimization runs, it occupies some time. In other words, the user is viewing a frame, at the mean time, the next incoming frame (the newest frame) is being processed simultaneously at the background. Any frames beyond the newest frame haven't been captured yet. The system is online as long as the processing time is less than the displaying time. For example, a video with framerate 30 fps. Its displaying time is 33.3 ms per frame. The processing time should be faster than 33.3 ms per frame for online performance. Curiously, if we enforce 0 frame latency, the processing time approaches 0, which is theoretically impossible. In practice, one frame latency is hard to be observed.

Equation 4 is quadratic. Similarly to Eq. 1, we optimize it by a Jacobi-based solver. The optimization is efficient. In our implementation, the buffer size is set to 40 frames. If a video has framerate about 30–50 fps, we roughly hold one second of past motions in the memory.

4.4 View Synthesis

After each optimization, the last frame in the buffer is warped towards the stabilized position, i.e. $\mathbf{P}^{(\xi)}(t)$, by a sparse update motion field for mesh vertices. The update motion is computed as, $\mathbf{U} = \mathbf{P} - \mathbf{C}$. Specifically, every vertex has an update motion vector. Therefore, we obtain an update motion mesh, i.e. a vector

per mesh vertex. The image content is warped according to the mesh warp [35]. Notably, the warping can be computed in parallel as well.

5 Discussions and Validation

In this section, we would like to validate the effectiveness of our proposed motion model in several aspects. When comparing with previous methods, we would like to exclude 3D methods [8,13,16] and single homography-based 2D methods [1,3, 20,21], because the former requires computationally expensive 3D reconstruction and is fragile for consumer videos while the latter can not represent spatially-variant motion and renders limited stability. Therefore, we focus on 2D or 2.5D methods which can handle spatially-variant motions. In general, these methods can be classified into three categories, smoothing long feature tracks [2,4,15], smoothing multiple 2D models [5,14,23] and smoothing dense flows [24].

If long feature tracks are provided (normally, average track length longer than 50 frames), the underlying 3D structure can be extracted from the tracks [2,19], and smoothing feature tracks gives strong results. However, long feature tracks are hard to obtain in consumer videos. Both the track length and the number of tracks drop quickly when there are quick camera swings, rotations or zoomings. For the robustness, our model does not rely on long feature tracks.

The second category proposes to smooth multiple parametric models estimated between neighboring frames for stabilization. The advantage is that they only require simple feature matches between two consecutive frames. Therefore, the robustness is largely improved. The drawback is that estimating these more advanced models are computationally expensive. They are not fast enough to achieve real-time performance.

The SteadyFlow belongs to the third category. It shows that the pixel profiles can well approximate the long feature tracks, which densely cover the entire video both spatially and temporally. While a feature track might start or end at any frame of the video, all pixel profiles begin at the first frame and end at the last frame, which is a much desired property for smoothing. Our MeshFlow resembles the SteadyFlow, and our vertex profiles resemble the pixel profiles. Our vertex profiles are a sparse version of pixel profiles. The SteadyFlow estimates pixel profiles by computing dense optical flow between neighboring frames. Then, the discontinuous motions in the flow, (e.g., motions on the boundary of different depth layers or on dynamic moving objects), are excluded by an iterative flow refinements, yielding a spatially smooth optical flow. We argue that the SteadyFlow overkills the problem. We can estimate sparse smooth flows by feature matches. Figure 7 shows a comparison between the SteadyFlow and the MeshFlow. We choose a dynamic video with a moving foreground. We show the raw optical flow calculated by [30] (Fig. 7(a)), the SteadyFlow (Fig. 7(b)) and our MeshFlow, interpolated into dense flow for visual comparison (Fig. 7(c)). As can be seen, our MeshFlow is quite similar to the SteadyFlow. The MeshFlow enjoys the merits of the SteadyFlow while the computation is much cheaper.

Fig. 7. (a) The raw optical flow calculated by [30]. (b) The SteadyFlow [24]. (c) Our MeshFlow interpolated into a dense field for visual comparison.

6 Experiments

We run our method on a laptop with 2.3 GHz CPU and 4G RAM. For frames with resolution 720×480, our un-optimized code can process a frame in 20 ms, without any acceleration of parallel computing. Specifically, we spend 6 ms, 1 ms, 10 ms and 3 ms to extract features, estimate MeshFlow, smooth vertex profiles and rendering frames, respectively.

6.1 Online Video Stabilization

We tried our method on various video sources. Figure 8 shows some examples. In each column, we show a capturing device and a sample video frame. The first example is a video captured by a general hand-held webcam. The second example is a micro UAV with real-time video transmissions. It often suffers from turbulence during the flight due to its small size and light weight, rendering videos with lots of vibrations and strong rolling shutter effects. The third example is a sport camera. It captures videos under wide view angles, which are transmitted through wifi in real-time. We show that our method can handle wide view angle lens as well. The last example is a tablet with the videos captured by its build-in camera. For all these examples, we show a side by side comparison which demonstrate original frames and the prompt stabilized results.

Fig. 8. Our experiments on various devices. Each column shows a capturing device and the corresponding sample frame.

6.2 Compare with Previous Methods

To compare our method with the previous methods, we slightly modify our programme to make it work on existing pre-captured videos. To imitate online processing, at each frame, our program reads in a frame from the video and processes it immediately, then saves it before processing the next frame. To evaluate the quality, we follow the approach of [5] which introduces three objective metrics: *cropping ratio*, *distortion* and *stability*. For a good result, these metrics should be close to 1. For completeness, we briefly introduce these metrics.

Cropping ratio measures the remaining area after cropping away black boundaries. A larger ratio means less cropping and hence better video quality. A homography B_t is fitted at each frame between input and output video. The cropping ratio for each frame can be extracted from the scale component of the homography. We average all ratios from all frames to yield the cropping ratio.

Distortion score is estimated from the anisotropic scaling of B_t, which can be computed by the ratio of the two largest eigenvalues of the affine part of B_t. The idea is borrowed for λ_t prediction in Sect. 4.2. Each frame has a distortion score, among which we choose the worst one as the final score for the whole video.

Stability score estimates the smoothness of the final video. Slightly different from the method in [5], we use the vertex profiles extracted from the stabilized video for evaluation. We analyze each vertex profile in the frequency domain. We take a few of the lowest frequencies (2nd to 6th) and calculate the energy percentage over full frequencies (DC component is excluded). Averaging from all profiles gives the final score.

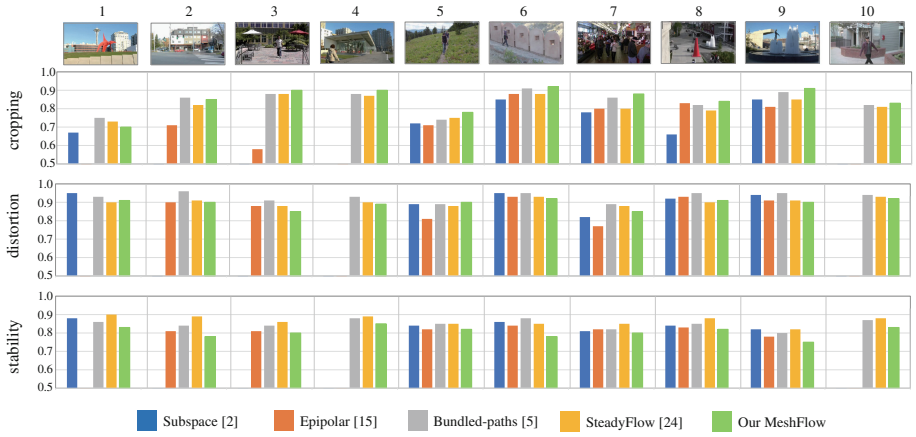

Fig. 9. Comparison with 10 publicly available videos in terms of three metrics.

We choose 10 publicly available videos and compare our method with the methods [2,5,15] and [24] in terms of the objective metrics. The result is reported in Fig. 9. The stabilized videos of these methods are either collected from their

project pages or provided by the corresponding authors. For videos that we do not find the result, we leave it blank.

It is slightly unfair for our method to compare with these offline approaches. We show that we can produce comparable quality. In general, our stability is slightly lower compared with the other methods. Because we only use the previous 40 frames for stabilization. We can buffer more previous frames for improved stability if they can fit into the memory. If some latency is allowed [9], we can even obtain future frames for improvements. The first several frames of the video have a relatively lower stability compared with other frames, as they are stabilized with a even smaller buffer. Specifically, the first frame is not stabilized and we begin to stabilize when there are at least three frames in the buffer. The stability increases gradually when more and more frames are hold in the buffer.

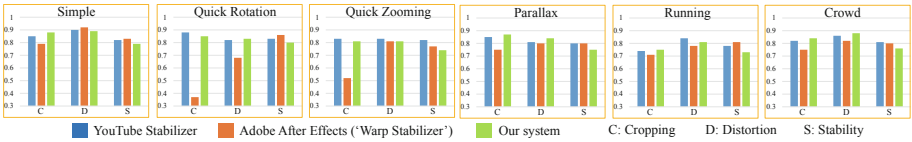

Fig. 10. Comparison with two offline video stabilization systems on datasets [5].

We further compare our method with two well-known commercial offline systems on a publicly available video data sets [5]. The two systems are Youtube Stabilizer developed according to methods [3,14] and Warp Stabilizer at Adobe After Effects built upon method [2]. The data sets group videos into several categories according to different scene types and camera motions, including (1) Simple, (2) Quick rotation, (3) Quick Zooming, (4) Large Parallax, (5) Crowd and (6) Running. The result is reported in Fig. 10. Similarly, it is not that fair to compare our method with these offline systems. We show our method is effective and robust to many challenging consumer videos.

7 Limitations

Our method can not handle videos containing large near-range foreground objects [36]. This is the common challenge faced by many previous methods [2,3,5,8,14,15,24]. Our method may also fail when features are insufficient for motion estimation under some extreme cases. Other types of features can be attempted for improvements [37,38].

8 Conclusions

We have presented a new motion model, *MeshFlow*, for online video stabilization. The MeshFlow is a sparse and spatially smooth motion field with motion vectors

only located at mesh vertices. By smoothing *vertex profiles*, motion vectors collected at mesh vertexes in MeshFlow over time, we can stabilize videos with spatially variant motion. Moreover, a *Predicted Adaptive Path Smoothing* (PAPS) is proposed to shift the method online with minimum latency. The experiment shows that our method is comparable with the state-off-the-art offline methods. The effectiveness is further validated by different capturing devices, which demonstrates potentials for practical applications.

Acknowledgements. This work is supported by National Nature Science Foundation of China (61502079 and 61370148). Ping Tan is supported by the NSERC Discovery Grant 31-611664, the NSERC Discovery Accelerator Supplement 31-611663.

References

1. Matsushita, Y., Ofek, E., Ge, W., Tang, X., Shum, H.: Full-frame video stabilization with motion inpainting. IEEE Trans. Pattern Anal. Mach. Intell. **28**, 1150–1163 (2006)
2. Liu, F., Gleicher, M., Wang, J., Jin, H., Agarwala, A.: Subspace video stabilization. ACM Trans. Graphics **30**(1), 4 (2011)
3. Grundmann, M., Kwatra, V., Essa, I.: Auto-directed video stabilization with robust l1 optimal camera paths. In: Proceedings of CVPR, pp. 225–232 (2011)
4. Wang, Y., Liu, F., Hsu, P., Lee, T.: Spatially and temporally optimized video stabilization. IEEE Trans. Vis. Comput. Graph. **19**, 1354–1361 (2013)
5. Liu, S., Yuan, L., Tan, P., Sun, J.: Bundled camera paths for video stabilization. ACM Trans. Graphics **32**(4), 78 (2013)
6. Karpenko, A., Jacobs, D.E., Baek, J., Levoy, M.: Digital video stabilization and rolling shutter correction using gyroscopes. In: Stanfor. Comput. Scie. Tech. Rep. CSTR 2011–03 (2011)
7. Bell, S., Troccoli, A., Pulli, K.: A non-linear filter for gyroscope-based video stabilization. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8692, pp. 294–308. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10593-2_20
8. Liu, F., Gleicher, M., Jin, H., Agarwala, A.: Content-preserving warps for 3D video stabilization. ACM Trans. Graphics **28**, 44 (2009)
9. Yang, J., Schonfeld, D., Chen, C., Mohamed, M.: Online video stabilization based on particle filters. In: Proceedings of ICIP, pp. 1545–1548 (2006)
10. Bae, J., Hwang, Y., Lim, J.: Semi-online video stabilization using probabilistic keyframe update and inter-keyframe motion smoothing. In: Proceedings of ICIP, pp. 5786–5790 (2014)
11. Jiang, W., Wu, Z., Wus, J., Yu, H.: One-pass video stabilization on mobile devices. In: Proceedings of Multimedia, pp. 817–820 (2014)
12. Liu, S., Wang, Y., Yuan, L., Bu, J., Tan, P., Sun, J.: Video stabilization with a depth camera. In: Proceedings of CVPR, pp. 89–95 (2012)
13. Zhou, Z., Jin, H., Ma, Y.: Plane-based content-preserving warps for video stabilization. In: Proceedings of CVPR, pp. 2299–2306 (2013)
14. Grundmann, M., Kwatra, V., Castro, D., Essa, I.: Calibration-free rolling shutter removal. In: Proceedings of ICCP, pp. 1–8 (2012)
15. Goldstein, A., Fattal, R.: Video stabilization using epipolar geometry. ACM Trans. Graphics **31**(5), 126 (2012)

16. Buehler, C., Bosse, M., McMillan, L.: Non-metric image-based rendering for video stabilization. In: Proceedings of CVPR, vol. 2, pp. 609–614 (2001)
17. Smith, B., Zhang, L., Jin, H., Agarwala, A.: Light field video stabilization. In: Proceedings of ICCV, pp. 341–348 (2009)
18. Ovrén, H., Forssén, P.E.: Gyroscope-based video stabilisation with auto-calibration. In: Proceedings of ICRA, pp. 2090–2097 (2015)
19. Liu, F., Niu, Y., Jin, H.: Joint subspace stabilization for stereoscopic video. In: Proceedings of ICCV, pp. 73–80 (2013)
20. Chen, B., Lee, K., Huang, W., Lin, J.: Capturing intention-based full-frame video stabilization. Comput. Graph. Forum **27**(7), 1805–1814 (2008)
21. Gleicher, M., Liu, F.: Re-cinematography: improving the camera dynamics of casual video. In: ACM Conference on Multimedia, pp. 27–36 (2007)
22. Morimoto, C., Chellappa, R.: Evaluation of image stabilization algorithms. In: Proceedings of ICASSP, vol. 5, pp. 2789–2792 (1998)
23. Bai, J., Agarwala, A., Agrawala, M., Ramamoorthi, R.: User-assisted video stabilization. Comput. Graph. Forum **33**(4), 61–70 (2014)
24. Liu, S., Yuan, L., Tan, P., Sun, J.: Steadyflow: spatially smooth optical flow for video stabilization. In: Proceedings of CVPR, pp. 4209–4216 (2014)
25. Trajković, M., Hedley, M.: Fast corner detection. Image Vis. Comput. **16**(2), 75–87 (1998)
26. Shi, J., Tomasi, C.: Good features to track. In: Proceedings of CVPR, pp. 593–600 (1994)
27. Zou, D., Tan, P.: Coslam: collaborative visual SLAM in dynamic environments. IEEE Trans. Pattern Anal. Mach. Intell. **35**(2), 354–366 (2013)
28. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proceedings of ISMAR, pp. 225–234 (2007)
29. Sun, D., Roth, S., Black, M.: Secrets of optical flow estimation and their principles. In: Proceedings of CVPR, pp. 2392–2399 (2010)
30. Liu, C.: Beyond pixels: exploring new representations and applications for motion analysis. Ph.D. thesis, MIT (2009)
31. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
32. Liang, C.K., Chang, L.W., Chen, H.H.: Analysis and compensation of rolling shutter effect. IEEE Trans. Image Process. **17**(8), 1323–1330 (2008)
33. Baker, S., Bennett, E., Kang, S.B., Szeliski, R.: Removing rolling shutter wobble. In: Proceedings of CVPR, pp. 2392–2399 (2010)
34. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, New York (2003)
35. Igarashi, T., Moscovich, T., Hughes, J.: As-rigid-as-possible shape manipulation. ACM Trans. Graph. **24**(3), 1134–1141 (2005)
36. Liu, S., Xu, B., Deng, C., Zhu, S., Zeng, B., Gabbouj, M.: A hybrid approach for near-range video stabilization. IEEE Trans. Circ. Syst. Video Technol. **PP**(99), 1 (2016). http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7457352
37. Zhou, H., Zou, D., Pei, L., Ying, R., Liu, P., Yu, W.: StructSLAM: visual SLAM with building structure lines. IEEE Trans. Veh. Technol. **64**(4), 1364–1375 (2015)
38. Li, S., Yuan, L., Sun, J., Quan, L.: Dual-feature warping-based motion model estimation. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 4283–4291 (2015)