First-person Hyper-lapse Videos

Johannes Kopf
Microsoft Research

Michael F. Cohen
Microsoft Research

Richard Szeliski
Microsoft Research

(a) Scene reconstruction (b) Proxy geometry (c) Stitched & blended

Figure 1: *Our system converts first-person videos into hyper-lapse summaries using a set of processing stages. (a) 3D camera and point cloud recovery, followed by smooth path planning; (b) 3D per-camera proxy estimation; (c) source frame selection, seam selection using a MRF, and Poisson blending.*

Abstract

We present a method for converting first-person videos, for example, captured with a helmet camera during activities such as rock climbing or bicycling, into *hyper-lapse* videos, i.e., time-lapse videos with a smoothly moving camera. At high speed-up rates, simple frame sub-sampling coupled with existing video stabilization methods does not work, because the erratic camera shake present in first-person videos is amplified by the speed-up. Our algorithm first reconstructs the 3D input camera path as well as dense, per-frame proxy geometries. We then optimize a novel camera path for the output video that passes near the input cameras while ensuring that the virtual camera looks in directions that can be rendered well from the input. Finally, we generate the novel smoothed, time-lapse video by rendering, stitching, and blending appropriately selected source frames for each output frame. We present a number of results for challenging videos that cannot be processed using traditional techniques.

CR Categories: I.3.8 [Computer Graphics]: Applications;

Keywords: hyper-lapse, time-lapse, image-based rendering

Links: DL PDF WEB VIDEO DATA CODE

1 Introduction

Yesterday's cameras were expensive, heavy, and difficult to operate devices, but those days are past. Today, digital cameras are cheap, small, easy to use, and have become practically ubiquitous. Cameras are now commonly attached to our cars, computers, phones and wearable cameras are becoming popular. Well known examples of wearable cameras include the GoPro, the Sony Action Cam, and Google Glass. We call these *first-person cameras*, since the action is seen as if through the eye of the camera operator.

First-person cameras are typically operated hands-free, allowing filming in previously impossible situations, for example during extreme sport activities such as surfing, skiing, climbing, or even sky diving. In many cases, first-person video is captured implicitly, rather than through explicit start-stop commands. Processing and consuming the resulting videos poses significant challenges for casual users. Such videos suffer from erratic camera shake and changing illumination conditions. More importantly, however, the videos are usually long and monotonous, which makes them boring to watch and difficult to navigate.

There has been a substantial amount of work on extracting important scenes from video [Chen et al. 2007; Detyniecki and Marsala 2008; Money and Agius 2008]. However, these techniques require high level scene understanding and have not reached a level of robustness that would make them practical for real-world tasks. A simpler and more robust technique that does not require scene understanding is time-lapse, i.e., increasing the speed of the video by selecting every n-th frame. Most first-person videos depict moving cameras (e.g., walking, hiking, climbing, running, skiing, train rides, etc.). Time-lapse versions of such videos are sometimes called *hyper-lapse* to emphasize that the camera is moving through space as well as accelerated through time.

Carefully controlled moving camera videos, such as those mounted on the front of a train, or extracted directional subsets of street-view panoramas, can be easily processed into hyper-lapse videos such as those at `http://hyperlapse.tllabs.io/` and `http://labs.teehanlax.com/project/hyperlapse`. Unfortunately, more casually captured videos such as from walking, running, climbing, or helmet mounted cameras during bicycling have significant shake and/or twists and turns. Increasing the frame rate of such videos amplifies the camera shake to the point of making these videos un-watchable.

Video stabilization algorithms could conceivably help create smoother hyper-lapse videos. Although there has been significant recent progress in video stabilization techniques (see Section 2), they do not perform well on casually captured hyper-lapse videos.

The dramatically increased camera shake makes it difficult to track the motion between successive frames. Also, since all methods operate on a single-frame-in-single-frame-out basis, they would require dramatic amounts of cropping. Applying the video stabilization *before* decimating frames also does not work because the methods use relatively short time windows, so the amount of smoothing is insufficient to achieve smooth hyper-lapse results.

In this paper, we present a method to create smooth hyper-lapse videos that can handle the significant motion noise in casually captured moving first-person videos. Similar to some previous stabilization techniques, we use structure-from-motion to build a 3D model of the world. We extend these techniques to a larger scale than previous work. We also compute per-frame 3D proxies for later use in our image-based rendering stage (Figure 1a–b).

The reconstructed camera positions plus geometric model allows us to optimize a new smoothed camera path in 6D pose space. Most previous methods employ relatively simple path smoothing algorithms, for example low-pass temporal filtering. With the substantial amount of smoothing required for time-lapse videos, simple algorithms do not give good results. In Section 5, we describe a novel optimization-based approach that balances several objectives.

Finally we render our output hyper-lapse video. Unlike previous stabilization work, we combine several input frames to form each output frame to avoid the need for over-cropping (see Figures 1c and 7). We leverage the recovered camera parameters and local world structure using image-based rendering techniques to reconstruct each output frame. We show resulting hyper-lapse videos from a number of input videos and discuss limitations and ideas for future work.

2 Previous Work

Our work is related to previous work in 3D video stabilization, 3D scene reconstruction, and image-based rendering.

Traditional video stabilization techniques use parameteric global transforms such as translations and rotations (optionally followed by local refinements), since these can be estimated quickly and robustly [Matsushita et al. 2006]. More recent techniques perform a full or partial (e.g., projective) reconstruction of the scene and camera motion. Liu et al. [2009] compute 3D camera trajectories and sparse 3D point clouds, which are then used to compute local "content-preserving" warps of the original video frames. In subsequent work, they extend their technique to work in cases where accurate 3D motion and geometry may not be available using subspace constraints on motion trajectories [Liu et al. 2011].

Grundmann et al. [2011] optimize the camera path based on L_1 norms of pose and its derivatives, which better simulate the actions of studio cameras. They also investigate the use of seam carving (video retargeting) techniques. Goldstein and Fattal [2012] use non-metric projective reconstruction and epipolar transfer to stabilize videos, again for cases where full 3D reconstructions cannot be reliably obtained, such as scenes with moving objects. Finally, Liu et al. [2013] use bundles of local camera paths to handle non-rigid effects such as rolling shutter while also minimizing geometric distortions. In our work, we also use 3D camera path reconstruction, but we synthesize our final frames from multiple source frames using image-based rendering.

In order to compute our 3D camera motions and scene proxies, we build upon techniques from structure-from-motion, which has seen dramatic progress in recent years [Snavely et al. 2006; Crandall et al. 2013; Wu 2013]. Similar to many of these papers, we use an incremental approach that adds well estimated cameras to the current 3D reconstruction. To handle our large datasets, we first remove redundant frames and then partition the frames into overlapping blocks. In order to estimate continuous 3D proxies for each frame, we interpolate a densified version of the 3D point cloud. Details on these components are presented in Section 4.

3 Overview

We captured first-person videos of several sport activities, such as cycling, hiking, or climbing, with GoPro Hero2 and Hero3 cameras (Table 1). The videos range from 3 to 13 minutes. Our goal is to speed these up by a factor of about $10\times$. As we show in Section 7, the amplification of camera shake resulting from the speed-up greatly exceeds the smoothing capabilities of existing video stabilization technologies.

The key to achieving better results with our system is in reconstructing an accurate representation of the scene, which allows us to optimize a new camera path in 6D pose space that is smooth while not deviating too far from the original cameras. It also supports using image-based rendering techniques and fusing multiple input frames to produce one output frame.

Our method consists of three stages:

1. **Scene reconstruction:** using structure-from-motion algorithms followed by dense depth map interpolation (Section 4);
2. **Path planning:** optimizing a 6D camera path that is smooth in location and orientation, passes near all of the input cameras, and is oriented towards directions which we can render well (Section 5);
3. **Image-based rendering:** projecting, stitching, and blending carefully selected input frames with per-frame proxy geometry (Section 6).

4 Scene Reconstruction

In the first stage of our system, we reconstruct both scene geometry and camera positions for each frame of the input.

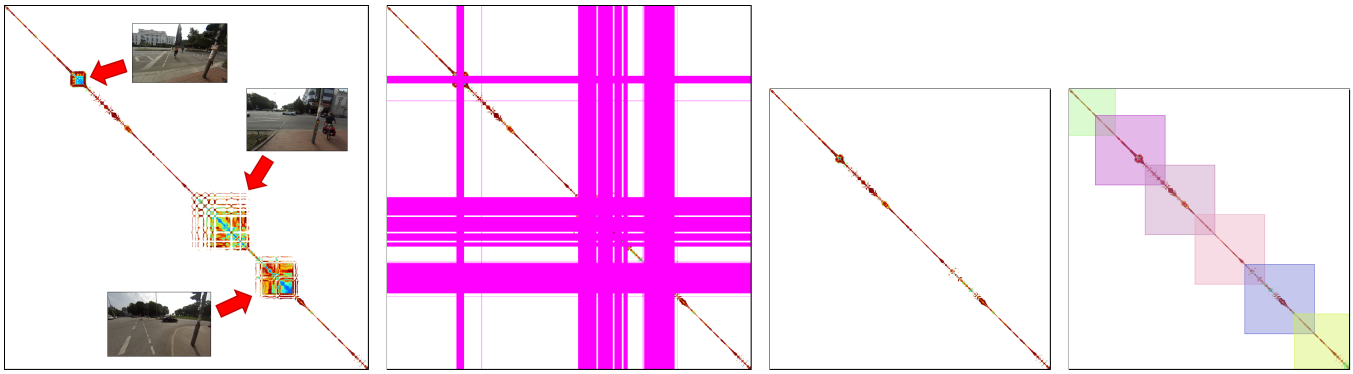
4.1 Preprocessing

The GoPro cameras we use capture video in a fish-eye projection with about 170° *diagonal* field of view. We use the OCamCalib toolbox [Scaramuzza et al. 2006] to calibrate the lens distortion and convert the videos to (cropped) linear perspective projection. While this change of projection is not strictly necessary, it simplifies the implementation of the remaining steps while removing only the most distorted corners of the fisheye images. The converted perspective input videos have a field of view of about $112^\circ \times 86^\circ$ in the horizontal and vertical directions, respectively.

4.2 Structure-from-Motion

Once these images have been reprojected, our next step is to estimate the extrinsic camera parameters of the input video frames as well as depth maps for the input images. This problem can be solved using structure-from-motion algorithms. We use an incremental algorithm similar to the ones described by Snavely et al. [2006] and Wu [2013]. The algorithm estimates the location and orientation of the input cameras. In addition, it computes a sparse 3D point cloud, where each point has an associated list of frames where it is visible.

The algorithm starts by finding feature points in every image and matching them among all pairs. For a small percentage of frames



(a) Original match graph (b) Redundant frames (c) Reduced match graph (d) Partition into batches

Figure 2: *Removing rendundant frames and batch processing. Places where the camera has stopped moving, e.g., at a red light, show up as blocks on the diagonal of the match graph. We detect and remove these, and partition the resulting reduced match graph into overlapping blocks for independent processing.*

that are strongly affected by motion blur or are completely textureless the algorithm might not find enough feature points that can be reliably matched. We drop these frames from the reconstruction and ignore them in the subsequent stages. Next, we remove redundant frames by searching for rows and columns in the match table that have large off-diagonal components and removing these from the original set of video frames (Figures 2a–c). We then run the remainder of the incremental structure-from-motion pipeline.

A difficulty with this approach is that incremental structure-from-motion algorithms do not scale well to problems as large as ours. In fact, we terminated an initial attempt to reconstruct a long video after a week of runtime. To alleviate this problem, we divide our datasets into overlapping batches of 1400 frames each with 400 frames overlap and reconstruct each batch separately in parallel (Figure 2d), which takes less than an hour for each batch.

To combine the batches into a single global coordinate system we first compute the best rigid alignment between all pairs using Horn's method [1987]. The cameras within the overlaps between batches now have 2 coordinates. We resolve the ambiguity by linearly combining them, where the blending weight moves from 0 to 1 in the overlap zone. Finally, we run one more round of bundle adjustment on the global system. After having obtained the full reconstruction, we scale it so that the average distance between two consecutive input cameras is 1. Figure 1a shows an example of the estimated camera path and sparse 3D point cloud produced by this pipeline.

4.3 Proxy Geometry

The reconstruction described in the previous subsection is sparse, since the depth is only known at a few isolated points in each frame. Accurately rendering the frame's geometry from a novel viewpoint requires dense depth maps or surface models. In this section, we describe how we use interpolation techniques to obtain smooth proxy geometries. To get the best quality, however, we need to first increase the density of points within each frame.

For this, we turn to *guided matching* [Beardsley et al. 1996]. Since we now have a good estimate of the camera poses, we can run feature point matching again but in a less conservative manner. Since a feature point in one image has to lie along the epipolar line in the neighboring images, we only match with other feature points nearby this line (we use a rather large search radius of 10 pixels to account for rolling shutter distortion). This dramatically increases the likelihood of finding matches.

As in other SfM algorithms, to robustly compute 3D points from feature matches, we form tracks of features across multiple frames. The original algorithm [Snavely et al. 2006] computes tracks by connecting all pairwise matching feature points with common endpoints. It then drops any tracks that loop back on themselves, i.e., that contain more than one feature point in the same image. We found this strategy too strict, since it forms and then rejects many large tracks. Instead we use a simple greedy algorithm that builds tracks by successively merging feature matches, but only if the merge would not result in a track that contains two features in the same image. Finally, we triangulate a 3D point for every track by minimizing the reprojection error. This is a non-linear least squares problem, which we solve using the Levenberg-Marquardt algorithm [Nocedal and Wright 2000].

Having increased the number of points, we are now ready to compute a dense mesh for every frame. We divide the field of view into a regular grid mesh of $w \times h$ vertices (we set $w = 41$ and h proportional to the inverse aspect ratio of the input video in our implementation). Our goal now is to compute the depth of every vertex in the mesh. However, since the reprojection error is related to disparity (inverse depth), we solve for disparity, $d(x)$, for every vertex x, instead. This is also practical because it avoids numerical problems with distant parts of the scene (i.e., points at infinity). Our objectives are to approximate the sparse points where the depth is known and to be smooth elsewhere. We achieve this by solving the following optimization problem:

$$\min_{\{d(x)\}} \sum_{x \in V} E_{depth}(x) + \sum_{x \in V,\ y \in N(x)} E_{smooth}(x,y), \tag{1}$$

where V is the set of vertices, and $N(x)$ is the 4-neighborhood of x.

The unary term

$$E_{depth}(x) = \sum_i B(x - p_i)\left(d(x) - z_i^{-1}\right)^2 \tag{2}$$

measures the approximation error. p_i and z_i are the image space projection and depth of the sparse reconstructed points, and B is a bilinear kernel whose width is one grid cell. The pairwise term

$$E_{smooth}(x,y) = \lambda\,(d(x) - d(y))^2 \tag{3}$$

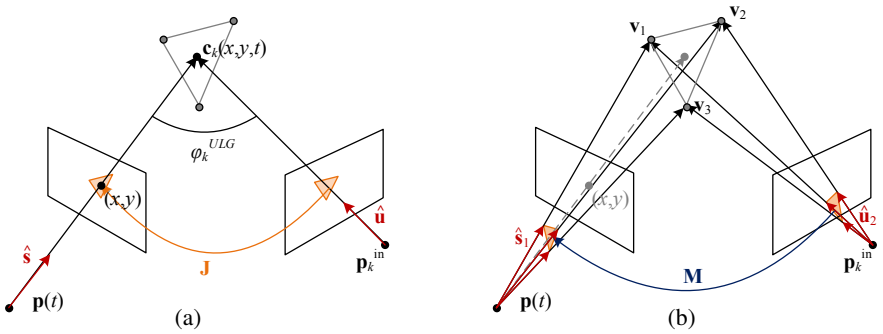


Figure 3: *Rendering quality term computations. (a) The Unstructured Lumigraph angular error φ_k^{ULG} measures the angle between the direction vectors $\hat{\mathbf{s}}$ and $\hat{\mathbf{u}}$ from the new $\mathbf{p}(t)$ and original $\mathbf{p}_k^{\text{in}}$ camera centers and the proxy surface point $\mathbf{c}_k(x,y,t)$. The affine transform $\mathbf{J}$ can be used to compute the texture stretch φ_k^{TS} at the reference pixel (x,y) but depends on the orientation of the new (rendering) camera. (b) The view invariant texture stretch φ_k^{TS3} can be computed using the condition number of the 3×3 mapping $\mathbf{M}$ between the unit vectors $\hat{\mathbf{s}}_i$ and $\hat{\mathbf{u}}_i$ pointing from the camera centers to the vertices $\mathbf{v}_i$ of the proxy triangle.*

encourages a smooth solution. $\lambda = 1$ balances between both objectives. The solution to Eq. 1 is a set of sparse linear equations, which we solve using a standard linear least-squares solver.

At this point, we have recovered the camera positions and orientations for each frame. In addition, we have a depth map in the form of a dense rectangular grid of points for each frame (Figure 1b).

5 Path Planning

Having a reconstruction of the scene, we now compute a smooth path for the output video. Our goal is to satisfy several conflicting objectives: we seek a path that is smooth everywhere and approximates the input camera path, i.e., it should not venture too far away from the input camera poses. Further it should be oriented toward directions that we can render well using image-based rendering.

Most previous methods employ path smoothing algorithms [Snavely et al. 2008; Liu et al. 2009; Liu et al. 2011; Goldstein and Fattal 2012; Liu et al. 2013]. However, we found that these simple algorithms cannot produce good results with our rather shaky input camera paths. We compare against these approaches in Section 7 and Figure 10.

Instead, we formulate path planning as an optimization problem that tries to simultaneously satisfy the following four objectives:

1. Length: The path should be no longer than necessary;

2. Smoothness: The path should be smooth both in position and orientation;

3. Approximation: The path should be near the input cameras;

4. Rendering quality: The path should have well estimated proxy geometry in view for image-based rendering.

We first formalize these objectives in Section 5.1, and then describe how to optimize them in Sections 5.2–5.5.

5.1 The Objectives

Let $\{\mathbf{p}_k^{\text{in}}, \mathbf{R}_k^{\text{in}}\}$ be the set of input camera positions and rotation matrices, and let $\mathbf{p}(t)$ and $\mathbf{f}(t)$ be the desired output camera continuous position and orientation curves, respectively. $\mathbf{f}(t)$ is represented as a unit front vector, i.e., it has two degrees of freedom. We define the remaining right and up vectors by taking cross products with a global world up vector. We assume the field of view of the output camera is a fixed user-supplied parameter and set it to 80% of the input camera's field of view in all our results.

The length and smoothness objectives are stated mathematically as penalty terms:

$$E_{length} = \int \left\|\mathbf{p}'(t)\right\|^2 \mathrm{d}t \tag{4}$$

$$E_{smooth\text{-}\mathbf{p}} = \int \left\|\mathbf{p}''(t)\right\|^2 \mathrm{d}t \tag{5}$$

$$E_{smooth\text{-}\mathbf{f}} = \int \left\|\mathbf{f}''(t)\right\|^2 \mathrm{d}t \tag{6}$$

For the approximation requirement, we seek to minimize the distance of every input camera to the closest point on the path:

$$E_{approx} = \sum_k \min_t \left\|\mathbf{p}_k^{\text{in}} - \mathbf{p}(t)\right\|^2 \tag{7}$$

The rendering quality is less straight forward, as it depends on the scene geometry as well the input camera positions. We seek to estimate the quality we can achieve using image-based rendering given a particular output camera position and orientation. Let $\varphi_k(x,y,t)$, defined below, be a penalty when using the geometry proxy of input camera k to render the pixel (x,y) at time t. The following expression then measures the expected quality for a particular output frame at time t by integrating over the image space:

$$\Phi(t) = \iint \min_k \varphi_k(x,y,t)\, \mathrm{d}x\,\mathrm{d}y. \tag{8}$$

We now integrate this penalty over the length of the curve to get our rendering quality penalty term:

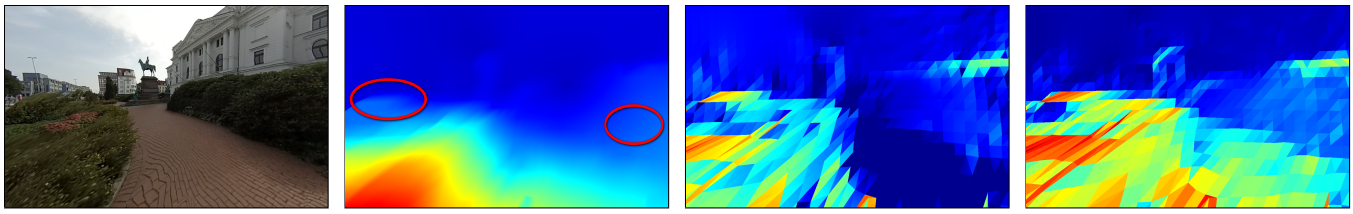
$$E_{quality} = \int \Phi(t)\, \mathrm{d}t. \tag{9}$$

How should φ_k be defined? Many possible choices have been discussed in the literature. In the Unstructured Lumigraph, Buehler et al. [2001] provide an overview, and propose using the angular error, i.e.:

$$\varphi_k^{ULG} = \cos^{-1}(\hat{\mathbf{s}} \cdot \hat{\mathbf{u}}), \tag{10}$$

where

$$\hat{\mathbf{s}} = \frac{\mathbf{c}_k(x,y,t) - \mathbf{p}_k^{\text{in}}}{\left\|\mathbf{c}_k(x,y,t) - \mathbf{p}_k^{\text{in}}\right\|}, \quad \hat{\mathbf{u}} = \frac{\mathbf{c}_k(x,y,t) - \mathbf{p}(t)}{\left\|\mathbf{c}_k(x,y,t) - \mathbf{p}(t)\right\|} \tag{11}$$



(a) Reprojected frame (b) Unstructured Lumigraph error (c) Texture stretch (d) Rotation invariant texture stretch

Figure 4: *Rendering quality term comparison. (b) The Unstructured Lumigraph error does does not always correlate well with visual artifacts; it has comparable values in the two encircled areas. However, in the reprojected frame, the left area shows strong texture stretching while the right one does not. (c-d) Directly measuring texture stretch correlates better with visual artifacts. Our new measure is invariant to view rotations.*

denote unit direction vectors between the camera centers and $\mathbf{c}_k(x,y,t)$, which denotes the intersection of the ray for pixel (x,y,t) with the geometry of the proxy for input camera k (Figure 3a). Unfortunately, this error does not take the obliqueness of the projection onto the proxy into account, and can associate very distorted views with low penalties (Figure 4b).

A better measure is to directly penalize the amount of texture stretch of the projection (Figure 4c):

$$\varphi_k^{TS} = 1 - \frac{\min_i \sigma_i^{\mathbf{J}}}{\max_i \sigma_i^{\mathbf{J}}}, \tag{12}$$

where $\sigma_i^{\mathbf{J}}$ are the singular values of the Jacobian of the texture coordinates:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}. \tag{13}$$

The Jacobian can be easily evaluated in a pixel shader using the `dFdx`/`dFdy` instructions, and the singular values for the 2x2 matrix computed using a closed form expression.

A disadvantage of this measure is that it is not invariant to the view orientation, since in a perspective projection, the periphery of the image is always more stretched than the center. It is preferable to have a measure that does not change as we rotate the viewpoint, since this allows for much more efficient optimization of the objective, as will become evident in Section 5.5.

We achieve this considering the stretch of *directions* (i.e., 3D unit vectors) rather then perspective projected 2D texture coordinates (Figure 4d). Let $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ be the vertices of a proxy triangle (Figure 3b). We now define the directions w.r.t. the input and output camera positions as:

$$\hat{\mathbf{s}}_i = \frac{\mathbf{v}_i - \mathbf{p}_k^{\text{in}}}{\left\|\mathbf{v}_i - \mathbf{p}_k^{\text{in}}\right\|}, \quad \hat{\mathbf{u}}_i = \frac{\mathbf{v}_i - \mathbf{p}(t)}{\|\mathbf{v}_i - \mathbf{p}(t)\|}. \tag{14}$$

This defines a linear mapping $\mathbf{M} = \mathbf{S}\mathbf{U}^{-1}$, where $\mathbf{S} = (\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2, \hat{\mathbf{s}}_3)$, $\mathbf{U} = (\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \hat{\mathbf{u}}_3)$. Our final IBR penalty function is thus:

$$\varphi_k^{TS3} = 1 - \frac{\min_i \sigma_i^{\mathbf{M}}}{\max_i \sigma_i^{\mathbf{M}}}, \tag{15}$$

where σ_i^M are the singular values of $\mathbf{M}$. We compute Eq. 15 in closed form in a shader (the source code is provided in the supplementary material).

The weighted sum of all these objectives gives our combined objective:

$$\begin{aligned} E = \lambda_1\, E_{length} + \lambda_2\, E_{smooth\text{-}\mathbf{p}} + \\ \lambda_3\, E_{smooth\text{-}\mathbf{f}} + \lambda_4\, E_{approx} + \lambda_5\, E_{quality}, \end{aligned} \tag{16}$$

where $\lambda_1 = 100$, $\lambda_2 = 100$, $\lambda_3 = 1000$, $\lambda_4 = 0.1$, $\lambda_5 = 0.01$ are balancing coefficients (recall that the scale of the reconstructed scene is normalized).

5.2 Optimization Strategy

Optimizing Eq. 16 directly is prohibitively expensive, since the $E_{quality}$ term is expensive to evaluate. It turns out, however, that we can greatly increase the tractability of the optimization by factoring it into two stages.

First, we optimize the location $\mathbf{p}(t)$ of the path while ignoring the energy terms that depend on the orientation $\mathbf{f}(t)$. While this reduced objective is still nonlinear, it can be efficiently optimized by iteratively solving sparse linear subproblems, as described in Section 5.3.

Next, we optimize the orientation $\mathbf{f}(t)$ of the output cameras while keeping the previously computed position curve $\mathbf{p}(t)$ fixed. This strategy dramatically improves efficiency because we designed our proxy penalty function φ_k^{TS3} to be rotation invariant. This enables us to precompute the min expression in Equation 8 once for all directions. We describe this part of the optimization in Section 5.5.

5.3 Optimizing the Path Location

In the first stage, our goal is to optimize the path location curve $\mathbf{p}(t)$ by minimizing the objectives E_{length}, $E_{smooth\text{-}\mathbf{p}}$, and E_{approx}, that do not depend on the orientation. We represent $\mathbf{p}(t)$ as a cubic B-spline curve, with the number of control vertices set to 5% of the number of input frames. The reduced objective now becomes:

$$\begin{aligned} E_{location} = \lambda_4 \sum_k \left\|\mathbf{p}_k^{\text{in}} - \mathbf{p}(t_k)\right\|^2 + \\ \lambda_1 \int \left\|\mathbf{p}'(t)\right\|^2 \mathrm{d}t \;+\; \lambda_2 \int \left\|\mathbf{p}''(t)\right\|^2 \mathrm{d}t, \end{aligned} \tag{17}$$

where $t_k = \arg\min_t \left\|\mathbf{p}_k^{\text{in}} - \mathbf{p}(t)\right\|$, is the parameter of the closest curve point to camera k. This is a standard spline fitting problem, which is frequently encountered in the literature.

While this is a non-linear objective, it can be efficiently solved using an iterative strategy. Note that the two integral terms in Eq. 17 have simple quadratic closed-form expressions for cubic B-splines. Now, the solution idea is to fix t_k during one iteration, which turns Eq. 17 into a quadratic problem that can be optimized by solving a sparse linear set of equations. The overall strategy is then to alternately optimize Eq. 17 and to update the t_k. Figure 5 shows a sample mapping of input frames to their corresponding t_k values while Figure 1a shows an example of a smoothed 3D camera path.

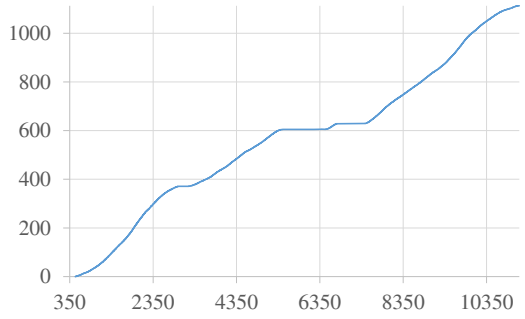


Figure 5: *Sample mapping from input frame numbers (horizontal axis) to their t_k values (vertical axis).*

For a detailed analysis of this algorithm and more implementation details, please see Wang et al.'s paper [2006].

5.4 Selecting Output Camera Positions Along Path

Having determined a continuous curve, $\mathbf{p}(t)$, that best meets our objectives aside from orientation, we now select camera positions along the curve. We now drop the parameter t and introduce the subscript i to refer to output frames. The curve samples are also our output video's frame positions.

A constant velocity along the curve in the hyper-lapse video is achieved by simply sampling of the curve into the desired number of output frames at evenly spaced locations along the curve in arc-length. Alternatively, if we wish to preserve some (or all) of the original camera velocities, we can use the mapping of input frames to their corresponding t_k values (Eq. 17) to compute a dynamic time warp. Sampling the curve in Figure 5 at regular (horizontal) intervals results in a set of non-uniformly spaced t samples that are denser in time when the original camera was slower or stopped.

In practice, we can blend between a constant and adaptive velocity. We show constant velocity results and an example of a variable velocity video in Section 7 and our supplementary materials.

5.5 Optimizing the Path Orientation

We now optimize the orientation curve $\mathbf{f}(t)$ by minimizing the $E_{smooth\text{-}\mathbf{f}}$ and $E_{quality}$ terms. The new objective becomes:

$$E_{orientation} = \lambda_5 \sum_i \Phi_i(\mathbf{f}_i) \; + \; \lambda_3 \sum_i \|2\mathbf{f}_i - \mathbf{f}_{i-1} - \mathbf{f}_{i+1}\|^2 . \qquad (18)$$

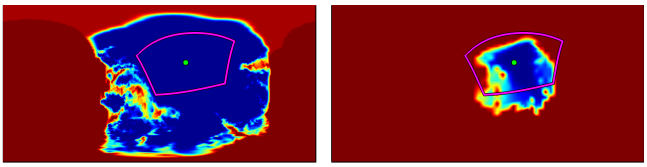
The key to making this optimization efficient lies in precomputing a $\Phi_i(\mathbf{f})$ lookup table for each output camera i, which we store in a cube map. First, at every output camera location $\mathbf{p}_i$, we render all proxy geometries using an appropriate shader (see supplementary material) and set the blending mode to compute the minimum in the frame buffer. Repeating this process in each of the six cardinal directions produces a cube map that stores:

$$\hat{\varphi}_i(\mathbf{f}) = \min_k \varphi_k(\mathbf{f}, i). \qquad (19)$$

Next, we compute the image integrals for the $\Phi_i(\mathbf{f})$ IBR fitness terms using these precomputed quantities:

$$\Phi_i(\mathbf{f}) = \iint_{I(\mathbf{f})} \hat{\varphi}_i(\mathbf{f}')\, d\mathbf{f}', \qquad (20)$$

where $I(\mathbf{f})$ indicates the set of rays $\mathbf{f}'$ that are in image I, and again store the results in cube maps (Figure 6). Since the functions in Equations 19 and 20 are relatively smooth, we use a cube face resolution of 64^2 and 16^2 pixels, respectively. This operation reduces evaluating the first term in Eq. 18 to a simple cube map texture fetch, and thus makes minimizing that equation extremely efficient. We use non-linear conjugate gradient with golden section line search to optimize it [Shewchuk 1994], and compute the partial derivatives of the cube map fetches analytically in a custom cube map sampler.



Per-pixel orientation penalty $\hat{\varphi}_i(\mathbf{f})$ Integrated orientation penalty $\Phi_i(\mathbf{f})$

Figure 6: *Look up tables for computing orientation penalties. The green dot shows the optimized forward vector* $\mathbf{f}$ *for one of the output frames, and the purple shape is the corresponding field of view. The orientation optimization algorithm tries to keep the orientation inside the feasible (dark blue) area of the second map.*

6 Rendering

Our final stage is to render images for the novel camera positions and orientations we discussed in the previous section. We use a greedy algorithm to select a subset of source images from which to assemble each output frame (Section 6.1). Then, we render each of the selected source images and stitch the results together using a Markov random field (MRF) and reconstruct the final images in the gradient domain (Section 6.2).

6.1 Source Frame Selection

For every output frame, our task is to determine a handful of input frames, which, when reprojected using their proxy geometry, cover the output frame's field-of-view with acceptable quality. We do this both for efficiency and to reduce popping, which might occur if each pixel were to be chosen from a different frame. For each output camera, we can trivially determine the nearest source camera. Unfortunately, that frame might point in a different direction than the output frame. For this reason we have to consider a relatively large range of input frames. In our implementation, we search over ± 500 frames around the nearest source camera. We start by rendering weight maps for each of these candidate frames, where:

$$w_{k,i}(x,y) = \text{clamp}\left(\frac{\varphi_{k,i}(x,y) - \tau_{min}}{\tau_{max} - \tau_{min}},\ 0,\ 1 \right) \qquad (21)$$

denotes a weight for using proxy k for output frame i. $\tau_{max} = 0.7$ is an upper threshold above which we consider the quality "good enough", and $\tau_{min} = 0.3$ is a lower threshold below which we deem the quality of the proxy too low to be used for rendering. For pixels that are not covered by the proxy, we set $w_{k,i} = 0$.

We now select the source frame that gives the highest overall quality:

$$s_0 = \arg\max_k \sum_{x,y} w_k(x,y). \qquad (22)$$

We keep selecting the images that give us the most *improvement* over the previously selected subset:

$$s_n = \arg\max_k \sum_{x,y} \max\left(0,\ w_k(x,y) - a_n(x,y)\right), \qquad (23)$$

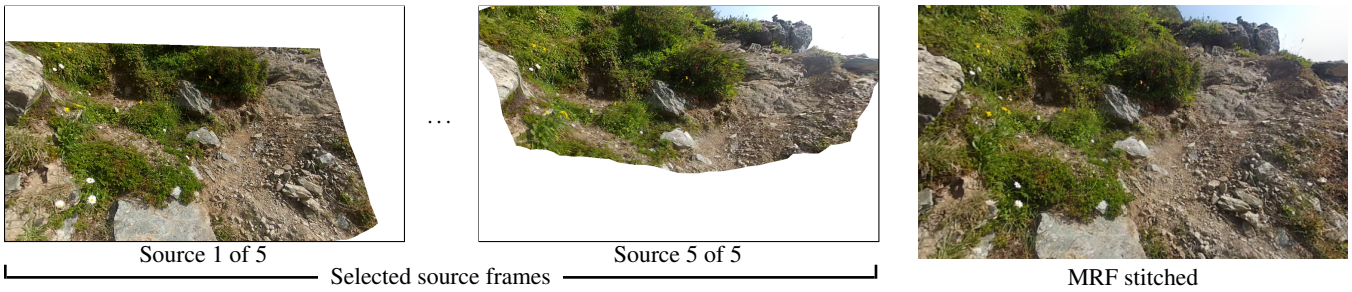


Figure 7: *Source frame selection and stitching. Left: source frame selection to ensure that all pixels are adequately covered with high-quality inputs; Right: Markov random field source pixel selection. (In practice, a spatio-temporal MRF is used—see Figure 1c.)*

where a_n is an accumulation buffer that contains the previously selected best value for every pixel:

$$a_n(x,y) = \max_{m<n} w_{s_m}(x,y). \tag{24}$$

We keep selecting source frames in this manner until the average improvement per pixel in Eq. 23 falls below a threshold of 0.1.

There are two more issues we need to address. First, some video frames are poor because of camera shake-induced motion blur. Let $b_k(x,y)$ be a per-pixel blur measure, which we obtain by low-pass filtering the gradient magnitude of the texture of image k. We now replace the weights used above by the following ones:

$$\tilde{w}_k = \frac{w_k b_k}{\max_l b_l}. \tag{25}$$

We also need to take the relative depths of pixels into account, to avoid selecting occluded parts of the scene. We render depth maps along with the weight maps, and for every pixel consider all the depth samples that fall onto it. We now want to discard all pixels that are occluded; however, we cannot use a strict z-buffer because we have to account for inaccuracies in the reconstructed depths.

Instead, we apply a Gaussian mixture model [Hastie et al. 2005] to these samples, where we determine the best number of Gaussians using the Bayesian information criterion [Schwarz 1978]. This essentially gives us a classification of the depths into one or several layers. We can now safely set the weights of every pixel not on the front layer to zero.

The above process selects on average 3 to 5 source frames for every output frame. While this is done independently for every output frame, we observe in practice that similar source frames are selected for nearby output frames. This is important for achieving more temporally coherent results when rendering. We encourage this even further by allowing every selected source to be not only used for the frame it was selected in, but also the surrounding ± 8 frames. However, in these extra frames we multiply the weight maps with a global attenuation coefficient that linearly drops to zero at the edges of the ± 8 frame window (indicated by the gray values in Figure 8). The attenuation tends to reduce popping artifacts in the stitching.

6.2 Fusion

Our last remaining task is to stitch and blend the previously selected source frames together to obtain the final output frames. We first optimize a discrete pixel labeling, where every (space-time) pixel p in the output video chooses the label α_p from one of the

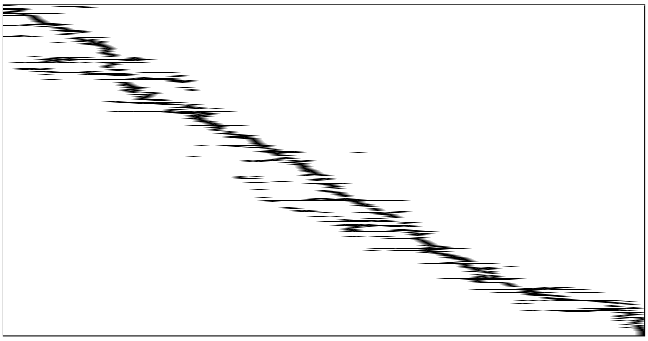


Figure 8: *Frame selection for the* CLIMBING *video. This image schematically shows (in black) which of the 331 source frames (rows) are used in each of the 636 output frames.*

rendered source proxies that have been selected for that particular frame [Agarwala et al. 2004]. We define the objective:

$$\min_{\{\alpha_p\}} \sum_p E_d(p,\alpha_p) + \lambda_{s\text{-}s} \sum_{p,q\in N(p)} E_s(p,q,\alpha_p,\alpha_q) + \lambda_{s\text{-}t} \sum_{p,q\in T(p)} E_s(p,q,\alpha_p,\alpha_q), \tag{26}$$

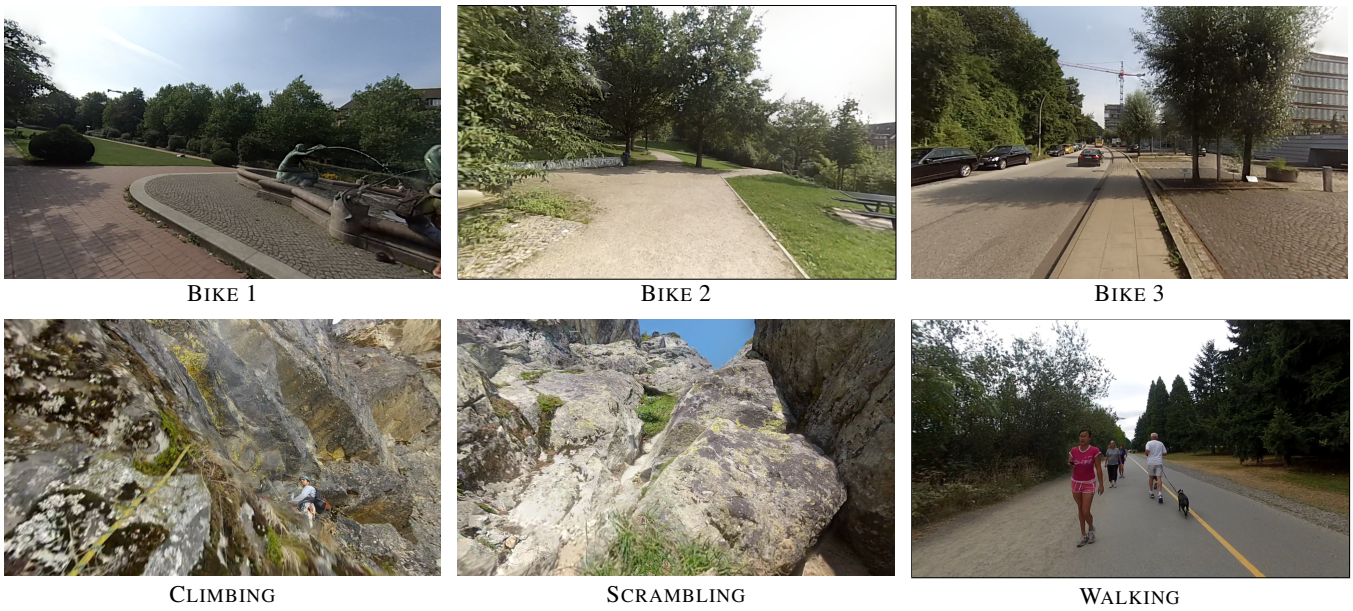
where the "data" term $E_d(p,\alpha_p) = 1 - w_{\alpha_p}(p)$ encourages selecting high quality pixels, the "smoothness" terms E_s, defined below, encourage invisible stitch seams, $\lambda_{s\text{-}s} = 10$, and $\lambda_{s\text{-}t} = 0.1$. $N(p)$ denotes the set of 4-neighbors within the same output frame, and $T(p)$ denotes the two temporal neighbors in the previous and next frames, which generally will not lie at the same pixel coordinates, and which we obtain by computing the medoid of the motion vectors of all candidate proxies at the given pixel).

Our smoothness terms are defined following previous work [Agarwala et al. 2004]:

$$E_s(p,q,\alpha_p,\alpha_q) = \left\|t_{\alpha_p}(p) - t_{\alpha_q}(p)\right\| + \left\|t_{\alpha_p}(q) - t_{\alpha_q}(q)\right\|, \tag{27}$$

where $t_{\alpha_p}(p)$ denotes the RGB value of the rendered proxy at pixel p. We solve Eq. 26 in a greedy fashion by successivly optimizing single frames while keeping the previously optimized frames fixed. Each frame's labels are optimized using the alpha expansion algorithm [Kolmogorov and Zabih 2004] in a coarse-to-fine manner [Lombaert et al. 2005].

The optimized labeling hides visible seams as much as possible. However, there might still be significant color differences because of exposure and white balancing changes in the source frames. We



BIKE 1 BIKE 2 BIKE 3

CLIMBING SCRAMBLING WALKING

Figure 9: *Sample rendered output frames from our six test videos*

Name	Duration mm:ss	Input Frames	Output Frames
BIKE 1	6:00	10,787	1111
BIKE 2	3:55	7,050	733
BIKE 3	13:11	23,700	2189
SCRAMBLING	9:07	16,400	1508
CLIMBING	3:37	6,495	636
WALKING	9:36	17,250	1744

Table 1: *Processed video statistics: activity and clip number, input duration (minutes and seconds), input frame count, output frame count.*

Stage	Computation time
Match graph (kd-tree)	10-20 minutes
Initial SfM reconstruction	1 hour (for a single batch)
Densification	1 hour (whole dataset)
Path optimization	a few seconds
IBR term precomputation	1-2 minutes
Orientation optimization	a few seconds
Source selection	1 min/frame (95% spent in GMM)
MRF stitching	1 hour
Poisson blending	15 minutes

Table 2: *Approximate computation times for various stages of the algorithm, for one of the longer sequences,* BIKE 3

balance out these differences by solving a spatio-temporal Poisson reconstruction problem to obtain our final pixels r:

$$\min_r \sum_p \lambda_{b\text{-}d}\big(r(p)-t_{\alpha_p}(p)\big)^2 + \lambda_{b\text{-}s}\Big(\big(\Delta_x r(p)-\Delta_x t_{\alpha_p}(p)\big)^2+\big(\Delta_y r(p)-\Delta_y t_{\alpha_p}(p)\big)^2\Big)+ \lambda_{b\text{-}t}\big(\Delta_i r(p)-\Delta_i t_{\alpha_p}(p)\big)^2, \tag{28}$$

where Δ_x, Δ_y, and Δ_i denote the horizontal, vertical, and temporal finite forward difference operator, respectively. $\lambda_{b\text{-}d}=0.001$, $\lambda_{b\text{-}s}=1$, and $\lambda_{b\text{-}t}=0.01$ are balancing coefficients. We solve Eq. 28 in a reduced domain using multi-splines with a spacing of 32 pixels [Szeliski et al. 2011] using a standard conjugate gradients solver [Shewchuk 1994].

7 Results and Evaluation

We present our experimental results on six video sequences acquired with GoPro Hero2 and Hero3 cameras. All of the videos were taken in a single shot and captured at 29.97 frames per second with a resolution of 1280×960 pixels. Table 1 lists the sequences, which are described by their activities and clip numbers, their input duration and frame number, and the number of output frames. The videos are between 3 and 13 minutes in length, and the output videos have a decimation (speed-up) factor of roughly $10\times$, a value that provides significant speed-up while still providing enough context to follow the motion.

Figure 9 shows a sample rendered output frame from each of the six videos. Our supplementary materials show the complete rendered videos (downsampled to 720 pixel width). We also show comparisons to more naïve methods to produce the same output length as our method. We include one rendered video where instead of using a constant camera velocity, we adapt the velocity to mimic that of the original camera motion, as described in Section 5.4.

As can be seen from these videos, our technique does an excellent job of providing fluid camera motion while minimizing rendering artifacts. The remaining artifacts are caused by errors in the proxy geometry due to nearby objects, independently moving objects, and wide changes in exposure, which the Poisson blend could not disguise. Some additional artifacts are due to sudden camera movements, errors in the reconstruction due to rolling shutter artifacts, and motion blur, despite our efforts to not use such frames for the rendering.

Computation Times. Several components of the hyper-lapse construction are computationally expensive, such as SfM reconstruction and spatio-temporal MRF stitching and Poisson blend-

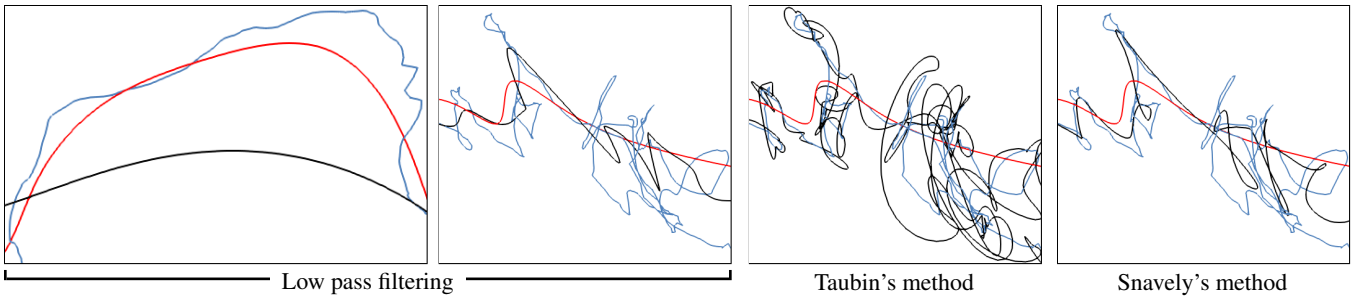


Figure 10: *Simple smoothing algorithms cannot produce satisfactory camera paths. The original camera path is shown in blue, our path in red, and the alternative path in black. Left: Low pass filtering the input camera positions produces a path that is over-smoothing in some part while it still contains sudden turns in other parts. Middle: Taubin's method [Taubin et al. 1996] runs into numerical problems. Right: The pull-back term in Snavely et al's algorithm [2008] prevents reaching a smooth configuration.*

ing. In this work, we were more interested in building a proof-of-concept system rather than optimizing for performance. As a result, our current implementation is slow. It is difficult to measure the exact timings, as we distributed parts of the SfM reconstruction and source selection to multiple machines on a cluster. Table 2 lists an informal summary of our computation times. We expect that substantial speedups are possible by replacing the incremental SfM reconstruction with a real-time SLAM system [Klein and Murray 2009], and finding a faster heuristic for preventing selecting of occluded scene parts in the source selection. We leave these speed-ups to future work.

7.1 Evaluating Alternatives

We experimented with a number of alternatives at each stage of the pipeline. Here, we outline some of those results.

Global 3D Model Reconstruction. We experimented with applying CMVS/PMVS [Furukawa and Ponce 2010], a state-of-the-art 3D reconstruction algorithm, to our input video and then rendering this model along our newly computed camera path. As can be seen in the `bike1_dense_recon.mp4` video, difficulties in reconstructing textureless and moving areas cause the algorithm to produce only a partial model, which cannot achieve the degree of realism we are aiming for with our hyper-lapse videos.

Path Planning. Some alternatives to our path planning results can be seen in Figure 10. As described in Section 5, we first experimented with low-pass smoothing the original camera poses. However, we found that there is no setting that reaches a good solution everywhere. In some regions, the path remains too noisy while in others, it oversmooths sharp turns with the same settings. Taubin's method [1996] is designed to smooth without the "shrinkage" problems. However, it runs into numerical problems before achieving sufficient smoothing. Snavely et al. [2008] use a variant of low-pass filtering that employs a pull-back term, which prevents the curve from moving too far away from the input curve. However, if this term is weighted too high, it prevents smoothing, while if it is weighted lower, it suffers from the same problems as regular low-pass filtering.

Video Stabilization. As mentioned in our introduction, we also experimented with traditional video stabilization techniques, applying the stabilization both before and after the naïve time-lapse frame decimation step. We tried several available algorithms, including the Warp Stabilizer in Adobe After Effects, Deshaker[1], and the Bundled Camera Paths method [Liu et al. 2013]. We found that they all produced very similar looking results and that neither variant (stabilizing before or after decimation) worked well, as demonstrated in our supplementary material. We also tried a more sophisticated temporal coarse-to-fine stabilization technique that stabilized the original video, then subsampled the frames in time by a small amount, and then repeated this process until the desired video length was reached. While this approach worked better than the previous two approaches (see the video), it still did not produce as smooth a path as the new technique developed in this paper, and significant distortion and wobble artifacts accumulated due to the repeated application of stabilization.

8 Conclusions

In this paper, we have shown how to create smooth hyper-lapse videos from casually captured first-person video. We leverage structure-from-motion methods to operate on very long sequences by clustering the input, solving a series of sub-problems, and merging results. We also densify the resulting point clouds in a second pass and then interpolate depth maps per input frame. This provides the input to a new path planning algorithm for the output hyper-lapse camera.

We have developed a new view-independent quality metric that accounts for foreshortening induced by texture-mapping source images onto final views. This novel metric is integrated into the path planning objective and results in a path that is both smooth and optimally placed and oriented to be renderable from the available input frames. Finally, each output frame is rendered from carefully selected source frames that are most capable of covering the frame as defined by the path planning. The source frames are stitched and blended to create the final frames. Our pipeline produces smooth hyper-lapse videos while limiting the need for severe cropping. The resulting videos could not be achieved by any existing method.

That said, there are many avenues to continue improving the results. We have generally relied on L_2 metrics for smoothness and stability. As Grundmann et al. [2011] showed, the use of L_1 measures often results in more natural stabilization results. Exploring such metrics throughout our process may result in even more natural feeling hyper-lapses. Also, the rolling shutter used in almost all modern video cameras causes a variety of *wobble* artifacts. Some recently reported work, such as Liu et al. [2013], should be applied to the input sequence before further processing. As stereo and structure-from-motion codes continue to improve, we hope to eliminate some

[1] `http://www.guthspot.se/video/deshaker.htm`

of the remaining artifacts caused by nearby surfaces, thin structures, and moving objects.

As the prevalence of first-person video grows, we expect to see a greater demand for creating informative summaries from the typically long video captures. Our hyper-lapse work is just one step forward. As better semantic understanding of the scene becomes available, either through improved recognition algorithms or through user input, we hope to incorporate such information, both to adjust the speed along the smoothed path, the camera orientation, or perhaps to simply jump over uninformative sections of the input. Finally, we look forward to being surprised by the many new and exciting videos of adventures recorded with first-person video systems.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Trans. Graph. 23*, 3, 294–302.

BEARDSLEY, P., TORR, P., AND ZISSERMAN, A. 1996. 3D model acquisition from extended image sequences. In *Fourth European Conference on Computer Vision (ECCV'96)*, Springer-Verlag, vol. 2, 683–695.

BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S., AND COHEN, M. 2001. Unstructured lumigraph rendering. *Proceedings of SIGGRAPH '01*, 425–432.

CHEN, F., COOPER, M., AND ADCOCK, J. 2007. Video summarization preserving dynamic content. In *Proceedings of the International Workshop on TRECVID Video Summarization*, ACM, New York, NY, USA, TVS '07, 40–44.

CRANDALL, D., OWENS, A., SNAVELY, N., AND HUTTENLOCHER, D. 2013. SfM with MRFs: Discrete-continuous optimization for large-scale reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 12, 2841–2853.

DETYNIECKI, M., AND MARSALA, C. 2008. Adaptive acceleration and shot stacking for video rushes summarization. In *Proceedings of the 2Nd ACM TRECVid Video Summarization Workshop*, ACM, New York, NY, USA, TVS '08, 109–113.

FURUKAWA, Y., AND PONCE, J. 2010. Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 8, 1362–1376.

GOLDSTEIN, A., AND FATTAL, R. 2012. Video stabilization using epipolar geometry. *ACM Trans. Graph. 32*, 5.

GRUNDMANN, M., KWATRA, V., AND ESSA, I. 2011. Auto-directed video stabilization with robust l1 optimal camera paths. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., AND FRANKLIN, J. 2005. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer 27*, 2, 83–85.

HORN, B. K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A 4*, 4, 629–642.

KLEIN, G., AND MURRAY, D. 2009. Parallel tracking and mapping on a camera phone. *International Symposium on Mixed and Augmented Reality (ISMAR 2009)*.

KOLMOGOROV, V., AND ZABIH, R. 2004. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 65–81.

LIU, F., GLEICHER, M., JIN, H., AND AGARWALA, A. 2009. Content-preserving warps for 3d video stabilization. *ACM Trans. Graph. 28*, 3, article no. 44.

LIU, F., GLEICHER, M., WANG, J., JIN, H., AND AGARWALA, A. 2011. Subspace video stabilization. *ACM Trans. Graph. 30*, 1 (Feb.), 4:1–4:10.

LIU, S., YUAN, L., TAN, P., AND SUN, J. 2013. Bundled camera paths for video stabilization. *ACM Transactions on Graphics (Proc. SIGGRAPH 2013) 32*, 4, article no. 78.

LOMBAERT, H., SUN, Y., GRADY, L., AND XU, C. 2005. A multilevel banded graph cuts method for fast image segmentation. *IEEE International Conference on Computer Vision (ICCV'05)*, 259–265.

MATSUSHITA, Y., OFEK, E., GE, W., TANG, X., AND SHUM, H.-Y. 2006. Full-frame video stabilization with motion inpainting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 7, 1150–1163.

MONEY, A. G., AND AGIUS, H. 2008. Video summarisation: A conceptual framework and survey of the state of the art. *Journal of Visual Communication and Image Representation 19*, 2, 121 – 143.

NOCEDAL, J., AND WRIGHT, S. J. 2000. *Numerical Optimization*. Springer.

SCARAMUZZA, D., MARTINELLI, A., AND SIEGWART, R. 2006. A toolbox for easily calibrating omnidirectional cameras. *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS 2006)*, 5695–5701.

SCHWARZ, G. 1978. Estimating the Dimension of a Model. *The Annals of Statistics 6*, 2, 461–464.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2006. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics (Proc. SIGGRAPH 2006) 25*, 3, 835–846.

SNAVELY, N., GARG, R., SEITZ, S. M., AND SZELISKI, R. 2008. Finding paths through the world's photos. *ACM Transactions on Graphics (Proc. SIGGRAPH 2008) 27*, 3, article no. 15.

SZELISKI, R., UYTTENDAELE, M., AND STEEDLY, D. 2011. Fast poisson blending using multi-splines. *International Conference on Computational Photography (ICCP 11)*.

TAUBIN, G., ZHANG, T., AND GOLUB, G. 1996. Optimal surface smoothing as filter design. In *Fourth European Conference on Computer Vision (ECCV'96)*, Springer-Verlag, vol. 1, 283–292.

WANG, W., POTTMANN, H., AND LIU, Y. 2006. Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Trans. Graph. 25*, 2, 214–238.

WU, C. 2013. Towards linear-time incremental structure from motion. In *International Conference on 3D Vision (3DV)*.