Gigapixel Panorama Video Loops

MINGMING HE, Hong Kong University of Science and Technology
JING LIAO, Hong Kong University of Science and Technology and Microsoft Research
PEDRO V. SANDER, Hong Kong University of Science and Technology
HUGUES HOPPE*, Microsoft Research

We present the first technique to create wide-angle, high-resolution looping panoramic videos. Starting with a 2D grid of registered videos acquired on a robotic mount, we formulate a combinatorial optimization to determine for each output pixel the source video and looping parameters that jointly maximize spatiotemporal consistency. This optimization is accelerated by reducing the set of source labels using a graph-coloring scheme. We parallelize the computation and implement it out-of-core by partitioning the domain along low-importance paths. The merged panorama is assembled using gradient-domain blending and stored as a hierarchy of video tiles. Finally, an interactive viewer adaptively preloads these tiles for responsive browsing and allows the user to interactively edit and improve local regions. We demonstrate these techniques on gigapixel-sized looping panoramas.

CCS Concepts: •**Computing methodologies** → **Computer graphics;** *Image manipulation; Computational photography;*

Additional Key Words and Phrases: video textures, cinemagraphs, video stitching

ACM Reference format:
Mingming He, Jing Liao, Pedro V. Sander, and Hugues Hoppe. YYYY. Gigapixel Panorama Video Loops. *ACM Trans. Graph.* VV, N, Article XXX (YYYY), 15 pages.
DOI: XXXXXXX.YYYYYYY

1 INTRODUCTION AND RELATED WORK

Forming image **panoramas** by assembling multiple photos from a shared viewpoint is a well-studied problem [Kopf et al. 2007; Szeliski and Shum 1997]. It involves registering the images based on their overlapping content [Szeliski 2006], finding good boundary seams [Agarwala et al. 2004; Summa et al. 2012], and merging the resulting regions to make the seams imperceptible [Agarwala 2007; Kazhdan and Hoppe 2008; Pérez et al. 2003]. For large domains, these computations can be parallelized and performed out-of-core [Kazhdan et al. 2010; Philip et al. 2011, 2015].

Much research also focuses on replacing traditional static photos by more dynamic representations such as **video loops**. The pioneering technique of video textures [Schödl et al. 2000] identifies similar frames in a short video to create natural loops. Graphcut textures [Kwatra et al. 2003] reduce spatiotemporal artifacts by allowing pixels to loop back at different times. Nonloopable scene regions can be selectively frozen, either with user guidance [Bai et al. 2012; Beck and Burg 2012; Joshi et al. 2012] or automatically [Bai et al. 2013; Tompkin et al. 2011]. Letting each pixel determine its own looping period provides additional flexibility to form good loops [Liao et al. 2013]. Finally, applying image morphing to segmented foreground objects enables video looping in the difficult case of a moving viewpoint [Sevilla-Lara et al. 2015].

Goal Our work lies at the confluence of these two research avenues: we reconstruct and render large panoramas with looping video content. An intermediate step in this direction is to embed video elements within a static panorama [Pirk et al. 2012; Tompkin et al. 2013]. Prior techniques for reconstructing fully looping panoramas use as input a video stream from a camera that rotates smoothly in a horizontal plane [Agarwala et al. 2005; Couture et al. 2011]. While this single continuous stream provides useful spatial coherence, the approach does not scale in the vertical direction. As with image panoramas, high-resolution content requires a more general traversal of the 2D space of directions.

Recent papers [Hermans et al. 2008; Rav-Acha et al. 2007] support more general types of videos and are not restricted to horizontal panning sequences. Rav-Acha et al. [2007] focus on the manipulation of chronological events for continuous time fronts rather than generating loops. Hermans et al. [2008] require foreground/background segmentation, and only retain repetitive dynamic elements on the background. Their patch-based algorithm does not compute per-pixel loops. All these methods are difficult to generalize to large-scale video panoramas. Perazzi et al. [2015] create large video panoramas by aligning and merging content from unstructured camera arrays with synchronously captured overlapping input videos. It should be feasible to compute video loops on such content. However, the approach involves a more expensive capture rig, consisting of 5 movie-grade cameras, for a panorama with at most 164M pixels.

Approach We capture a 2D grid of partially overlapping videos using a single consumer camera affixed to a motorized tripod mount, and stitch these unsynchronized videos into a large-scale dynamic panorama. After projectively aligning the videos, we globally optimize a video loop for the entire panoramic scene. As illustrated in Fig. 2, this processing involves several stages including video stabilization, vignetting correction, gain compensation, loop optimization, and gradient-domain Poisson blending.

Contributions The processing pipeline includes the following main technical contributions:

- We propose what to our knowledge is the first practical method to create gigapixel-sized looping panoramas. The method is able to achieve high-quality results at extremely high resolution, while keeping a reasonable cost in memory and time.

*Now at Google Inc.

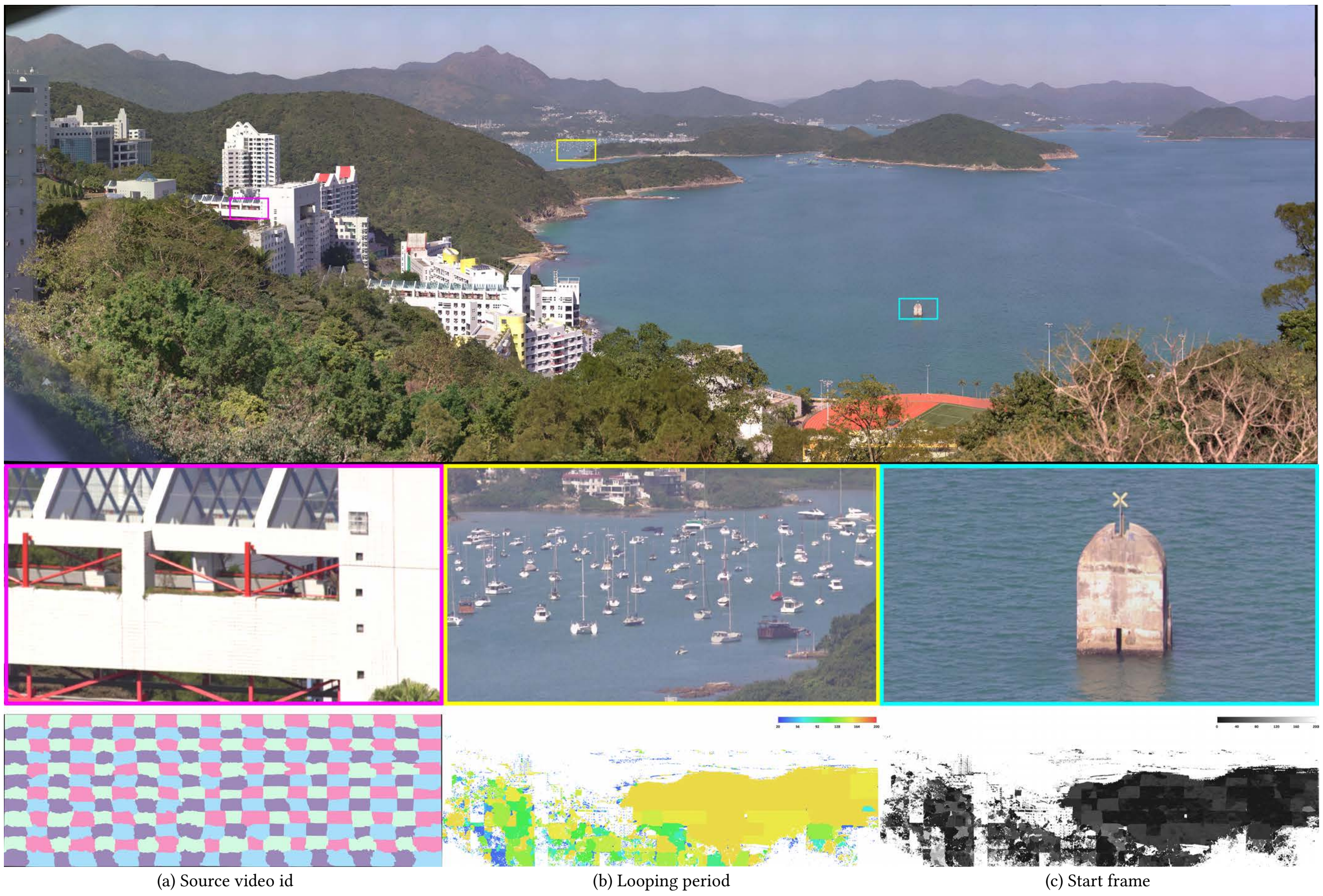


Fig. 1. We synthesize a large panoramic looping video by spatiotemporally stitching content from a 2D grid of overlapping source videos. The top row shows one frame of the resulting panorama video loop, with highlighted close-ups in the middle row. A combinatorial optimization minimizes spatial seams and temporal pops by selecting at each pixel the best source video, looping period, and start frame, as visualized in the three corresponding subfigures in the lower row. The source video indices are visualized in (a) with different colors, the looping periods are shown in (b) with increasing periods from blue to red, and the start frames are shown in (c), where brighter colors indicate later frames. Pixels that are *static* are shown in white in (b) and (c).

- We present a loop optimization framework which optimally retrieves and stitches content from overlapping unsynchronized videos in the spatiotemporal domain to generate seamlessly looping panoramic videos. The approach also considers motion consistency and dynamism preservation to improve the visual quality.
- We develop a series of novel optimization strategies tailored for our method to improve performance, including a k-coloring technique that accelerates optimization by reducing the number of labels, and a parallelized out-of-core solver that builds a content-adapted domain decomposition.
- We introduce an algorithm for interactively editing the video panorama results using our tile-based viewer.

Overview We next present the successive stages of the processing pipeline as illustrated by Fig. 2. Starting with the captured video data (Sec. 2), several preprocessing steps are applied, including stabilization and gain compensation (Sec. 3), followed by panorama alignment and further color correction (Sec. 4). Our out-of-core loop optimization then generates looping content for the entire panorama (Sec. 5), and assembles it with gradient-domain Poisson blending (Sec. 6). The looping gigapixel video is diced into square tiles, used by our efficient gigapixel video renderer (Sec. 7). Finally, we perform experiments and analysis on various challenging gigapixel panorama cases to demonstrate the efficacy and robustness of our method (Sec. 8).

2 VIDEO CAPTURE

We use a GigaPan Pro robotic arm to automatically sweep through a 2D grid, capturing spatially overlapping 8-second video segments in column-major order. Spatially adjacent videos require at least a 20% overlap to allow accurate stitching and looping optimization. The GigaPan is designed to act as a remote shutter trigger and

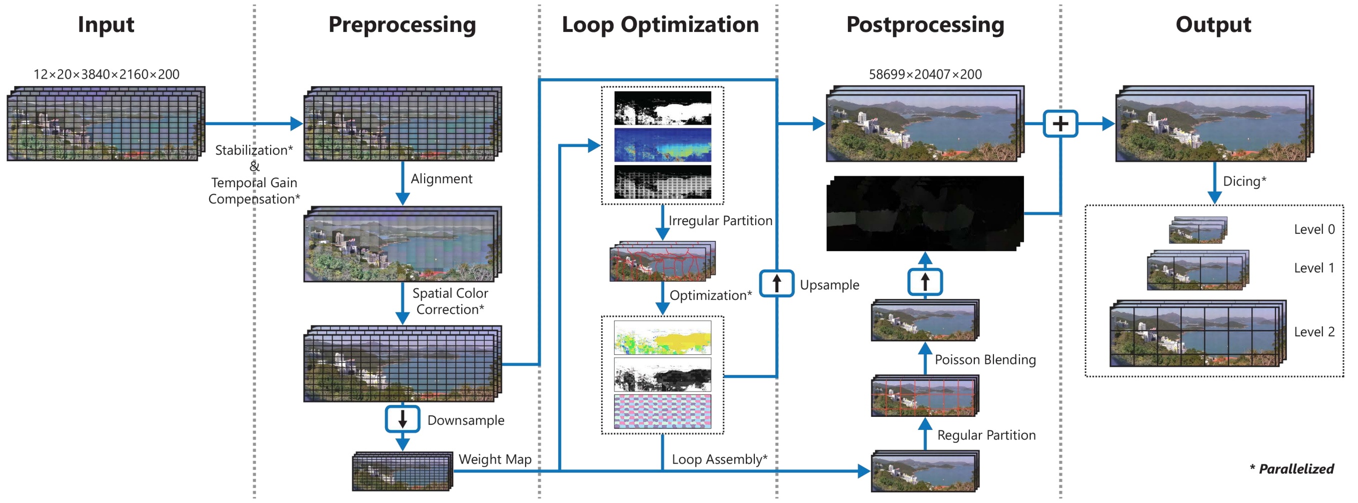


Fig. 2. Stages of our processing pipeline for creating a gigapixel panorama video loop. The input consists of 240 partially overlapping 8-second video segments, each at 25fps and 4K resolution. The output is a hierarchy of 8-second video tiles covering a gigapixel (58,699×20,407) panorama domain. First, we preprocess the input videos to obtain a panorama video by correcting brightness temporally, aligning the videos, and harmonizing color spatially. Next, the panorama video is downsampled for optimization. During optimization, it is partitioned into several regions for efficient out-of-core processing. Each region is then optimized individually. Screened Poisson blending removes spatial seams and temporal pops during postprocessing. The problem involves gathering sparse gradient differences in the downsampled loop, solving a Poisson problem out-of-core, upsampling the resulting correction, and merging with the initial high-resolution loop. Finally, the output loop is diced into regular video tiles at multiple scale levels for efficient rendering.

automatically take images at each position. We disable the shutter trigger feature and instead capture a single rolling video of the entire process. By configuring the GigaPan to use a 13-second "exposure", we ensure that the camera remains for at least 13 seconds in each position. We then extract 8 seconds of video within this 13-second window for each position in the rolling video. In practice, we have found that 8-second video segments suffice to create our loopable videos. However, these segments can be easily adjusted to shorter or longer.

3 VIDEO SEGMENT PREPROCESSING

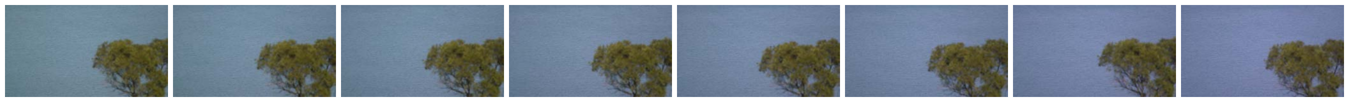
Video stabilization Although the captured video data is largely stable, the large zoom factor occasionally leads to minor instability in the presence of strong wind gusts, even with a robust, heavy-duty tripod. Thus, to form a seamless video loop, the input videos first need to be stabilized. In our experiments, a simple stabilization solution based on 2D affine transformation is sufficient to generate visually stable results for most cases. Between each pair of adjacent frames, we calculate optical flow using the iterative Lucas-Kanade method [Bouguet 2001] and estimate an optimal global affine transformation for stabilization. Taking the start frame as the reference frame, we calculate the relative camera motion transformation of each frame by integrating all previous affine transformations. After applying the transformations, frames may have missing data near the boundaries. We address this by cropping the videos accordingly. Since the overlap region between neighboring videos is large, all video segments can still be aligned and assembled without any gaps. Please refer to the accompanying video for a demonstration.

Temporal gain compensation All the video segments are captured using automatic exposure settings. Significant changes in brightness may occur due to metering, since the camera may need time to adjust the exposure at each new position. Such variations in brightness make it difficult to generate a long looping result. To address this issue, we include a temporal gain compensation step to estimate a gain factor per color channel for every frame. Based on our experiments, a simple solution that adjusts the brightness of each frame to match the overall average pixel brightness of a representative video frame (in our case, the average of the last 10 frames) can well address this problem. Fig. 3 shows the importance of applying temporal gain compensation to achieve a temporally stable result that can be used to generate loops.

4 PANORAMA ALIGNMENT AND COLOR CORRECTION

Median frame computation Once the video segments are stabilized, the frames within each video segment should correspond to the same camera pose. The next goal is to determine the relative alignment of the different segments. For this alignment, we apply techniques from panorama image stitching, The main hurdle is to identify a single representative frame from each video segment. Ideally, we would like it to be a relatively stationary frame, with no dynamic, moving objects which could hinder the alignment quality, as shown in Fig. 4. To achieve this, for each segment, we compute the median value of each pixel over all the frames, thus removing the color value outliers caused by moving objects.

Alignment Given the representative median frames, we can apply an off-the-shelf image panorama stitching technique to estimate camera parameters for each video segment. This technique is used to align all the frames of our input videos into a gigapixel rectangular grid. In our implementation, we used the camera parameters estimated by Microsoft Image Composite Editor [Microsoft Research


(a) video frames before temporal gain compensation

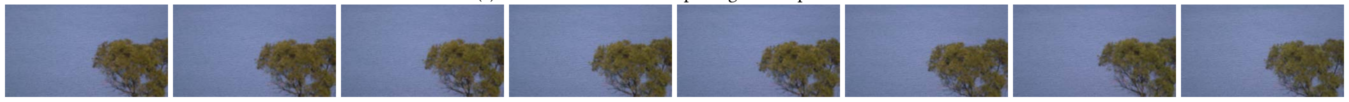
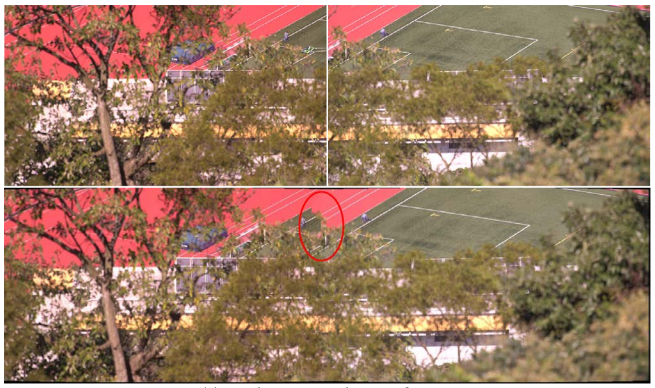

(b) video frames after temporal gain compensation

Fig. 3. Automated processing in the camera can cause temporal changes in brightness and white-balance within each input video. Enabling good video loops requires compensating for these changes.


(a) stitching two arbitrary frames

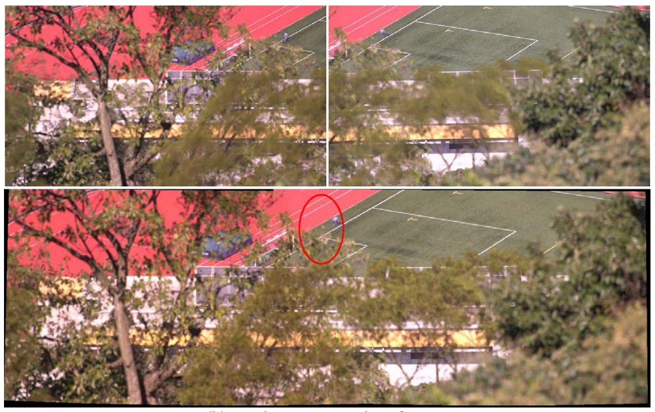

(b) stitching two median frames

Fig. 4. When arbitrary frames are used for inter-video alignment (top row), dynamic objects that are not present in both frames can hinder stitching quality (second row). By instead using median frames (third row), such troublesome objects are removed or blurred, thus avoiding potentially mismatched feature points and reducing the chances of stitching problems (bottom row).

2016]. Note that all the frames in each post-stabilized video segment share the same camera parameters.

Devignetting and spatial color correction Spatial color correction is important to reduce artifacts at segment boundaries. Lens vignetting causes a reduction of brightness at the periphery of each segment. If uncorrected, this vignetting leads to an obvious repetitive pattern in the stitched panorama, as evidenced in Fig. 5a. We apply the method of [Goldman 2010] to address vignetting. A separate concern is that videos captured at different times with distinct photometric settings may lead to obvious lighting differences, which adversely effects stitching and loop generation. To create seamless loops, the overall brightness should not change significantly across segments and frames, so harmonizing video brightness across neighbors is also necessary. To address this issue, following the method of [Brown and Lowe 2007], we compute spatial gain factors for R, G, and B channels separately based on the median frames and apply these to all the video segments. Fig. 5 shows the importance of applying devignetting and color correction in order to achieve a seamless result.

5 LOOPING OPTIMIZATION

Given the 2D grid of aligned overlapping videos, we select source content at each panorama pixel to form a spatiotemporally consistent video loop.

5.1 Basic optimization framework

We formulate our optimization algorithm following the basic framework of Liao et al. [2013], which optimizes a loop given a *single* input video.

Given an input video with color $V(x, t_i)$ within the range $[0, 255]$ per color channel at each 2D pixel x and input frame time t_i, the aim is to compute a video loop

$$L(x, t) = V(x, \phi(x, t)),\ 0 \leq t < T, \tag{1}$$

by determining a time-mapping function ϕ defined from unknown start time s_x and looping period p_x at each pixel as

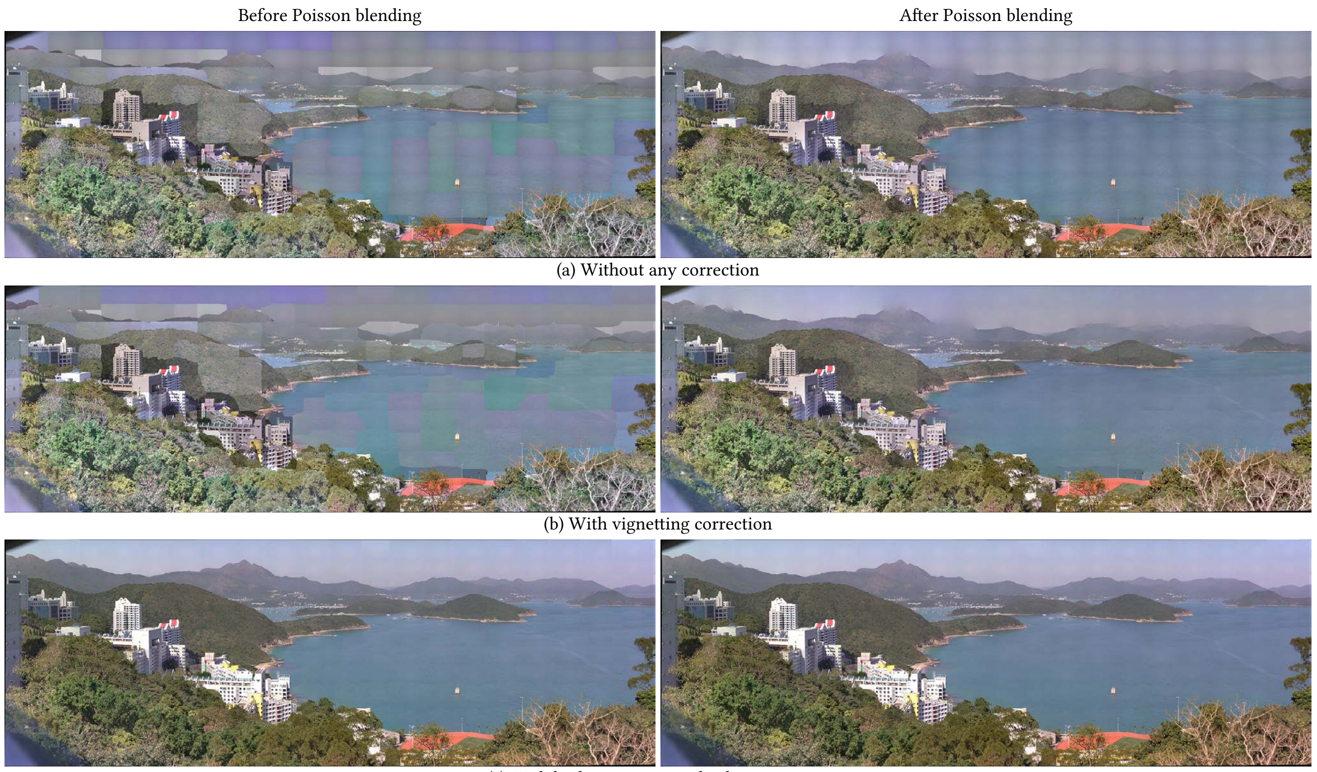
$$\phi(x, t) = s_x + ((t - s_x) \bmod p_x) \tag{2}$$

Note that the looping content $L(x, \cdot)$ at position x is always taken from input video $V(x, \cdot)$ at the same location (Fig. 6). Pixels in nonloopable regions may be assigned with a fixed period $p_x = 1$, to make them static by freezing their color to that in frame s_x.

The goal is to determine the set of periods $\mathbf{p} = \{p_x\}$ and start frames $\mathbf{s} = \{s_x\}$ that minimize the objective

$$E'(\mathbf{s}, \mathbf{p}) = \beta_{\text{spatial}} E'_{\text{spatial}}(\mathbf{s}, \mathbf{p}) + E'_{\text{temporal}}(\mathbf{s}, \mathbf{p}) + \beta_{\text{static}} E'_{\text{static}}(\mathbf{s}, \mathbf{p}), \tag{3}$$

where the constant weight β_{spatial} balances the spatial and temporal terms and the weight β_{static} penalizes static pixels. (We set $\beta_{\text{spatial}} = 7$ and $\beta_{\text{static}} = 200$ as default.)



(a) Without any correction

(b) With vignetting correction

(c) With both vignetting and color correction

Fig. 5. Alignment results with and without vignetting correction and spatial color correction are shown on the left images. The images on the right show the final results after Poisson blending (performed after loop optimization as described in Section 6). Note that devignetting, color correction, and Poisson blending are all necessary to achieve a seamless result.

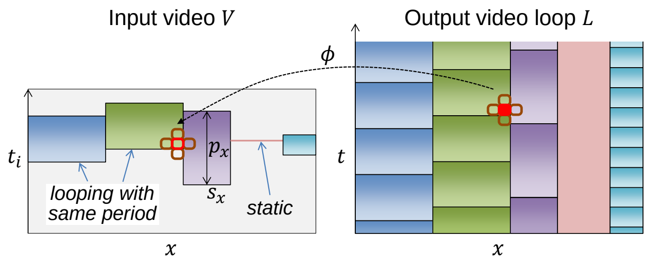


Fig. 6. A video loop L is formed from an input video V by repeating some temporal interval at each pixel x using a time-mapping function ϕ, specified using a period p_x and start frame s_x. The consistency objective is that for any output pixel color (shown in solid red), its spatiotemporal neighbors should have the same values as the corresponding neighbors in the input [Liao et al. 2013].

The first two terms E'_{spatial} and E'_{temporal} together measure the spatiotemporal consistency of neighboring colors in the video loop with respect to the input video [Agarwala et al. 2005]. Respectively, E'_{spatial} integrates consistency over all spatially adjacent pixels (x, z):

$$E'_{\text{spatial}}(\mathbf{s}, \mathbf{p}) = \sum_{\|x-z\|=1} \Psi_{\text{spatial}}(x, z)\gamma_s(x, z) \text{ with} \tag{4}$$

$$\Psi_{\text{spatial}}(x, z) = \frac{1}{T}\sum_{t=0}^{T-1}\begin{pmatrix}\|V(x, \phi(x, t)) - V(x, \phi(z, t))\|^2 + \\ \|V(z, \phi(x, t)) - V(z, \phi(z, t))\|^2\end{pmatrix},$$

and E'_{temporal} integrates consistency across the two loop end frames s_x and $s_x + p_x$ at each pixel:

$$E'_{\text{temporal}}(\mathbf{s}, \mathbf{p}) = \sum_x \begin{pmatrix}\|V(x, s_x) - V(x, s_x + p_x)\|^2 + \\ \|V(x, s_x - 1) - V(x, s_x + p_x - 1)\|^2\end{pmatrix}\gamma_t(x). \tag{5}$$

Specifically, E'_{temporal} compares the loop start/end frames with the frames immediately after/before the loop to measure consistency across the transition point. The modulation factors γ_s and γ_t attenuate consistency in high-frequency regions similar to Kwatra et al. [2003] and Bai et al. [2012], but are further supplemented with a blend mask $B(x)$ as in Liao et al. [2015]:

$$\begin{aligned}\gamma_s(x, z) &= 1/(1 + \lambda_s \text{MAD}_{t_i}\|V(x, t_i) - V(z, t_i)\|)\max(B(x), B(z)) \\ \gamma_t(x) &= 1/(1 + \lambda_t \text{MAD}_{t_i}\|V(x, t_i) - V(x, t_i + 1)\|)B(x),\end{aligned} \tag{6}$$

where MAD is the temporal median absolute deviation of color difference and $B(x) = \text{clamp}(\max_{t_i}(c_b V(x, t_i + 1) - V(x, t_i)), 0, 1)$ with the scaling factor $c_b = 0.006$. (We set $\lambda_s = 0.490$ and $\lambda_t = 1.634$ in our results.)

Finally, the static term E'_{static} is introduced to prevent a trivial all-static solution:

$$E'_{\text{static}}(\mathbf{s},\mathbf{p}) = \sum_{x|p_x=1} \text{MAD}_{t_i} \|V(x,t_i) - V(x,t_i+1)\|. \tag{7}$$

Minimizing E' is a 2D Markov Random Field (MRF) problem, in which each pixel is assigned a label (p_x, s_x) among the outer product $\{p\} \otimes \{s\}$ of candidate periods and start frames. An approximate solution can be found using an iterative multilabel graph cut algorithm [Kolmogorov and Zabih 2004].

Liao et al. [2015] accelerate the optimization by masking out non-loopable pixels from 2D graph cut, pruning graph-cut labels based on dominant periods, and optimizing on a coarse grid while retaining finer detail. We also adopt these techniques which together reduce computation times about two orders of magnitude. As illustrated in Fig. 2, we first compute a spatiotemporally downsampled version of the input video using a 3D box filter, with both temporal and spatial scaling factors set to 4. For example, one input video with resolution $3840 \times 2160 \times 200$ is downscaled to $960 \times 540 \times 50$. Looping optimization will be performed only on the downsampled video. During the post-processing step after optimization, we first upsample the optimized label map to the input resolution by nearest interpolation to reconstruct the initial loop L. To further remove its spatiotemporal inconsistencies, we then generate the Poisson-blended loop L' using gradient-domain (Poisson) blending. Due to the prohibitive computational cost of solving the large linear system at the original resolution, we choose to perform Poisson blending on the downsampled loop $\hat{L}$, and upsample the Poisson-blending difference $\hat{L_d} = \hat{L}' - \hat{L}$ with a box filter to get L_d at the original resolution. Finally, L_d is added to the initial loop L to obtain the blended loop L'.

5.2 Improvements to optimization framework

Focusing on our goal of panorama video looping, we further augment the basic optimization framework to overcome various challenging issues commonly seen in panorama videos. Specifically, we propose to preserve consistent motion and encourage local dynamism to achieve vivacious yet consistent and natural results, as reflected in our new objective function, which consists of an improved spatial consistency term E_{spatial}, a new dynamism preserving term E_{dynamic}, and the aforementioned terms $E_{\text{temporal}} = E'_{\text{temporal}}$, $E_{\text{static}} = E'_{\text{static}}$:

$$\begin{aligned} E(\mathbf{s},\mathbf{p}) = E_{\text{temporal}}(\mathbf{s},\mathbf{p}) + \beta_{\text{spatial}} E_{\text{spatial}}(\mathbf{s},\mathbf{p}) + \\ \beta_{\text{dynamic}} E_{\text{dynamic}}(\mathbf{s},\mathbf{p}) + \beta_{\text{static}} E_{\text{static}}(\mathbf{s},\mathbf{p}). \end{aligned} \tag{8}$$

Next, we describe E_{spatial} and E_{dynamic}.

Motion consistency One drawback of prior work is that only color consistency is considered in the spatial term. As demonstrated in the red circle of Fig. 7b, noticeable artifacts often arise at the boundaries between static and moving elements, which suggests that motion consistency is also critical, especially for wide-angle scenes. To address this issue, we consider motion consistency by making use of the alpha (α) color channel to store a measure of local motion magnitude within the video:

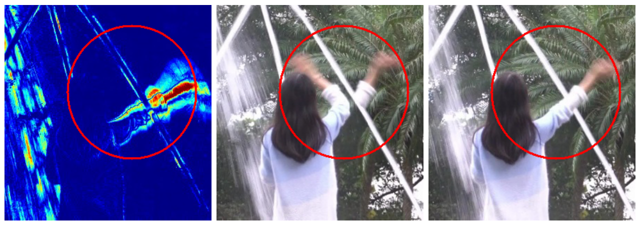


Fig. 7. We augment the input video with an additional channel V^α that measures motion, resulting in loops with better spatial consistency.

$$V^\alpha(x,t_i) = \sqrt{\sum_{c\in\{R,G,B\}} \left(V^c(x,t_i) - \frac{\sum_{t=t_i-n}^{t_i+n} V^c(x,t)}{(2n+1)} \right)^2}. \tag{9}$$

It uses the color difference between the pixel x at frame t and its temporal neighborhood within a $2n+1$ window around t ($n = 2$ in our experiments). Including this component can greatly improve spatial motion consistency. Motion magnitude is assigned with an independent constant weight c_{motion} (we set $c_{\text{motion}} = 2$ for all our results). The term E_{spatial} is now defined as:

$$E_{\text{spatial}}(\mathbf{s},\mathbf{p}) = \sum_{\|x-z\|=1} \Psi_{\text{spatial}}(x,z)\gamma_s(x,z) + c_{\text{motion}} \Psi_{\text{motion}}(x,z), \tag{10}$$

with

$$\Psi_{\text{motion}}(x,z) = \frac{1}{T} \sum_{t=0}^{T-1} \begin{pmatrix} \|V^\alpha(x,\phi(x,t)) - V^\alpha(x,\phi(z,t))\|^2 + \\ \|V^\alpha(z,\phi(x,t)) - V^\alpha(z,\phi(z,t))\|^2 \end{pmatrix}. \tag{11}$$

A direct comparison with and without this motion consistency term is shown in Fig. 7. This term also helps the stitching seam between multiple videos to go through regions with similar motion consistency, as seen in Fig. 8c.

Dynamism preservation When optimizing multiple videos with overlapping content, we find that both temporal and spatial terms tend to favor less dynamic content in overlapping regions across videos that have unequal degrees of dynamism. To address this, we introduce a novel dynamic energy term:

$$E_{\text{dynamic}} = -\sum_x \sum_{t=s_x}^{p_x+s_x-1} \left\| V(x,\phi(x,t)) - V(x,\phi(x,t-1)) \right\|^2. \tag{12}$$

This term measures the variance of the video content within the selected looping content, and further encourages looping content with greater dynamism (higher variance). If the dynamism weight β_{dynamic} in Eq. (8) is low, dynamic regions tend to be frozen or eliminated and the whole scene may lack vitality; whereas if it is too high, some nonloopable motions may appear unnatural. Experimentally, we suggest a value in $[0, 0.3]$, depending on the input content and user preference. If the input video is a large-scale scene with only subtle motions (Fig. 14(top)), we set $\beta_{\text{dynamic}} = 0.3$ to encourage more dynamism, while if too many chaotic moving objects are present (Fig. 16(bottom)), we decrease β_{dynamic} to 0 to reduce non-loopable behavior and improve temporal consistency.

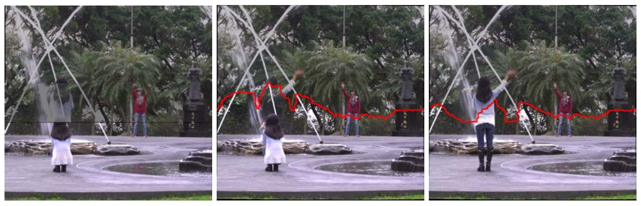


Fig. 8. Comparison of looping results when assigning video indices using (b) panorama image stitching and using (c) our integrated optimization.

5.3 Generalization to multiple input videos

Our looping optimization extends the aforesaid single-input framework to multiple overlapping input videos $\{V_i\}$. The key to this problem is to determine for any overlapped pixel x the index i_x of the corresponding video V_{i_x} from which this pixel is to be retrieved. This is equivalent to finding a series of stitching seams corresponding to the discontinuities of this index map.

A naïve approach to determine these seams would be to apply a traditional panorama stitching method to those representative video frames. However, this would be problematic due to the poor spatial consistency of the subsequently chosen looping content, resulting in obvious stitching artifacts (Fig. 8b).

Therefore, we instead propose to integrate this stitching step directly into the combinatorial looping optimization, to obtain much improved spatial consistency (Fig. 8c). Our solution is to include the video index i_x into the set of unknowns to be solved at each pixel. Thus, the loop definition becomes

$$L(x,t) = V_{i_x}(x, \phi(x,t)),\ 0 \le t < T. \tag{13}$$

We then optimize the video indices $\mathbf{i} = \{i_x\}$, periods $\mathbf{p} = \{p_x\}$, and start frames $\mathbf{s} = \{s_x\}$ to minimize the energy with generalized temporal consistency term $E_{\text{temporal}}(\mathbf{i},\mathbf{s},\mathbf{p})$, spatial consistency term $E_{\text{spatial}}(\mathbf{i},\mathbf{s},\mathbf{p})$, and dynamism preserving term $E_{\text{dynamic}}(\mathbf{i},\mathbf{s},\mathbf{p})$:

$$\begin{aligned} E(\mathbf{i},\mathbf{s},\mathbf{p}) = E_{\text{temporal}}(\mathbf{i},\mathbf{s},\mathbf{p}) + \beta_{\text{spatial}}E_{\text{spatial}}(\mathbf{i},\mathbf{s},\mathbf{p}) + \\ \beta_{\text{dynamic}}E_{\text{dynamic}}(\mathbf{i},\mathbf{s},\mathbf{p}) + \beta_{\text{static}}E_{\text{static}}(\mathbf{s},\mathbf{p}). \end{aligned} \tag{14}$$

To achieve good spatial consistency across video boundaries, the loop should avoid undesirable spatial seams due to inconsistently overlapping videos. For each pair of adjacent pixels x and z with different video indices i_x and i_z, our goal is to minimize this visual inconsistency at both locations. Specifically, this is measured as pixel differences $\|V_{i_x}(x,\cdot) - V_{i_z}(x,\cdot)\|$ at x and $\|V_{i_x}(z,\cdot) - V_{i_z}(z,\cdot)\|$ at z respectively. Note that we avoid comparing $V_{i_x}(x,\cdot)$ with $V_{i_z}(z,\cdot)\|$ directly to let each pixel's neighbors look consistent as in the input videos, instead of simply enforcing color similarity. This way, the spatial term E_{spatial} adopts a modified spatial color consistency function Ψ^*_{spatial}:

$$\begin{aligned} &\Psi^*_{\text{spatial}}(i,x,z) = \\ &\frac{1}{T}\sum_{t=0}^{T-1}\left(\begin{array}{c} \left\|V_{i_x}(x,\phi(x,t)) - V_{i_z}(x,\phi(z,t))\right\|^2 + \\ \left\|V_{i_x}(z,\phi(x,t)) - V_{i_z}(z,\phi(z,t))\right\|^2 \end{array} \right) \gamma_\kappa(i_x,i_z), \end{aligned} \tag{15}$$

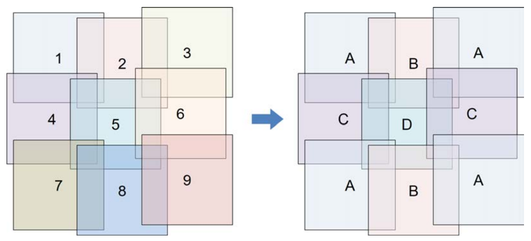


Fig. 9. The grid structure is such that no domain pixel is covered by more than four input videos, so we remap video index labels to just four values (here, colors labeled A, B, C, and D).

where the *video source coherence factor* $\gamma_\kappa(i_x,i_z)$ ($\kappa = 2$ for all our results) is defined as:

$$\gamma_\kappa(i_x,i_z) = \begin{cases} 1 & \text{if } i_x = i_z, \\ \kappa & \text{if } i_x \neq i_z, \end{cases} \tag{16}$$

which gives greater importance to spatial consistency at the stitching boundaries between different source videos, based on the observation that these inter-video boundaries are much more likely to have visible artifacts than intra-video seams due to scene motion and the larger time difference in the acquired content. Intuitively, we encourage these boundaries to go through regions that are more compatible.

Both the temporal term E_{temporal} and the dynamic term E_{dynamic} can be adapted from Eq. (5) and Eq. (12) respectively, by using V_{i_x} in place of V. If a particular video index i_x is invalid for a pixel x because the pixel lies outside the video extent, we set the spatial and temporal terms to infinity to prevent the index from being selected.

Graph-coloring acceleration We minimize the objective function Eq. (14) using a multilabel graph cut, in which the set of pixel labels $\{(i,p,s)\}$ is the outer product of candidate input video indices $\{i\}$, periods $\{p\}$, and start frames $\{s\}$. Thus the number of candidates is

$$|\{(i,p,s)\}| \propto |\{i\}| \times |\{p\}| \times |\{s\}|. \tag{17}$$

Compared to the single-video case, the introduction of video indices significantly increases the number of candidate labels. This is a concern due to the linear relationship between optimization time and the number of labels.

We assume video segments are on a 2D grid with less than 50% overlapping areas between arbitrary adjacent segments, so that each pixel will be covered by no more than four video segments. According to this layout pattern of the input videos, we can reuse the video indices for different sets of non-overlapping videos. The number of index labels cannot exceed the maximum number of videos that overlap at a given region. Due to our regular-grid video pattern, we are able to reduce the labels to just 4 as shown in Fig. 9.

We apply a similar strategy to prune the number of candidate periods. The optimization allows pixels to have any looping period but it is too costly if all periods $\{p\}$ and all possible start frames $\{s\}$ for each period are considered. For one single 4× temporally downsampled 8-second input video (i.e., 50 frames) and a minimum loop period of 8 frames, this results in a multilabel graph cut with 953 labels ($s = 0,1,\ldots 49|p = 1; s = 0,1,\ldots 42|p = 8; s = 0,1,\ldots 41|p =$

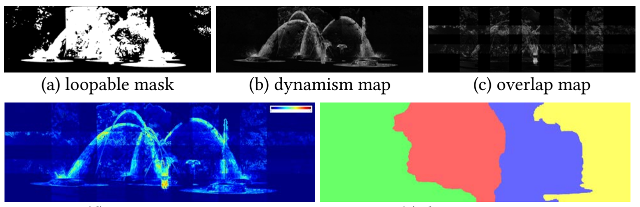


Fig. 10. The content-adapted domain partition places boundaries near pixels that are either nonloopable or have low dynamism.

9; ...; $s = 0|p = 50$). To reduce the number of labels for the single-video setting, Liao et al. [2015] find that two dominant periods are usually sufficient to generate a good looping result. With multiple videos, our experiments suggest that it is sufficient to restrict the candidate periods at any pixel to be the union of all dominant periods from the 3×3 nearest input videos (one video and its eight neighbors). Thus, similar to 4-coloring for video indices, we use a 9-coloring pattern to reuse period labels. Here each color represents two dominant periods. Thus with the addition of a static period label (period $p = 1$), the optimization needs only consider at most 19 candidate period labels.

With these two graph-coloring accelerations, the upper bound of candidate labels becomes

$$|\{(i, p, s)\}| \propto 4 \times 19 \times |\{s\}|, \tag{18}$$

which is independent of the number of input videos.

For inputs with other video layouts where more than 50% adjacent video overlap may occur, we can no longer apply this graph-coloring scheme to reduce the number of labels. But this problem can be simply bypassed by performing video cropping to enforce proper overlapping ratio. Our method can also potentially be extended to arbitrarily positioned tiles that do not lie on a 2D grid. In that case, a more general coloring strategy must be employed to assign the index labels avoiding repetition among neighbors. As a result, the optimization may involve a larger number of index labels and thus be potentially more expensive.

Out-of-core loop optimization Due to the large amount of data required for a gigapixel video, we solve the loop optimization out-of-core. Although many implementations of distributed graph cuts (e.g., Strandmark and Kahl [2010], Liu and Sun [2010]) have been proposed, their constraints on optimization and memory prevent us from applying them directly to our situation. We design a scalable algorithm by spatially partitioning the panorama into multiple regions that are as independent as possible and solve each region separately. The pixels from adjacent already-solved regions serve as boundary conditions when solving a region. We parallelize the computation of non-adjacent regions on separate machines.

Although the region boundary constraints provide spatial consistency, these can hinder the quality of the solution. In particular, the order of optimized regions affects the final result. To mitigate this order dependence, ideally we would like the partition boundaries to pass through pixels which are masked out of the optimization. Liao et al. [2015] compute a binary loopable classification mask which excludes pixels that are either static or cannot possibly form a good loop. We let our loopable mask loopable(x) be the union of the loopable masks of all video segments (Fig. 10a). Note that a perfect partition through non-loopable pixels may not exist. If a partition boundary must pass through loopable pixels, we seek to minimize the amount of dynamism along that boundary. This is because pixels that exhibit little dynamism can usually have any potential artifacts addressed by Poisson blending. We define the dynamism of each pixel as dynamic$(x) = \max_{i,t} \left\|V_i^{\alpha}(x,t)\right\|$, as visualized in Fig. 10b. In the multi-video case, another factor that must be considered is the overlap. The partition cannot avoid going through the overlap region but we prefer it to pass overlapping pixels that are similar among all video sources, since for those pixels the video ID selection is not critical. We define an overlap function as overlap$(x) = \max_{i,j,t} \left\|V_i(x,t) - V_j(x,t)\right\|$, as visualized in (Fig. 10c). Jointly considering these three factors (loopable, dynamism and overlap), the importance of each pixel is defined as

$$\text{impor}(x) = (\text{loopable}(x) * \text{dynamic}(x)) + \text{overlap}(x), \tag{19}$$

where loopable(x) is Boolean function (implicitly converted to 0 or 1) while dynamic(x) and overlap(x) are normalized between 0 and 1. The importance map is shown in Fig. 10d.

The partition can be seen as a weighted, 4-connected graph partition problem. Given a desired number k of regions, the optimized partition $\mathbf{P} = P_1, P_2, ...P_k$ minimizes the sum of edge cut costs

$$d(x, z) = \min(\text{impor}(x), \text{impor}(z)) \tag{20}$$

over all adjacent pixels x, z mapped to different partition regions.

To make the regions roughly uniform in size, we add a constraint on the area $A(P_i)$ of each region:

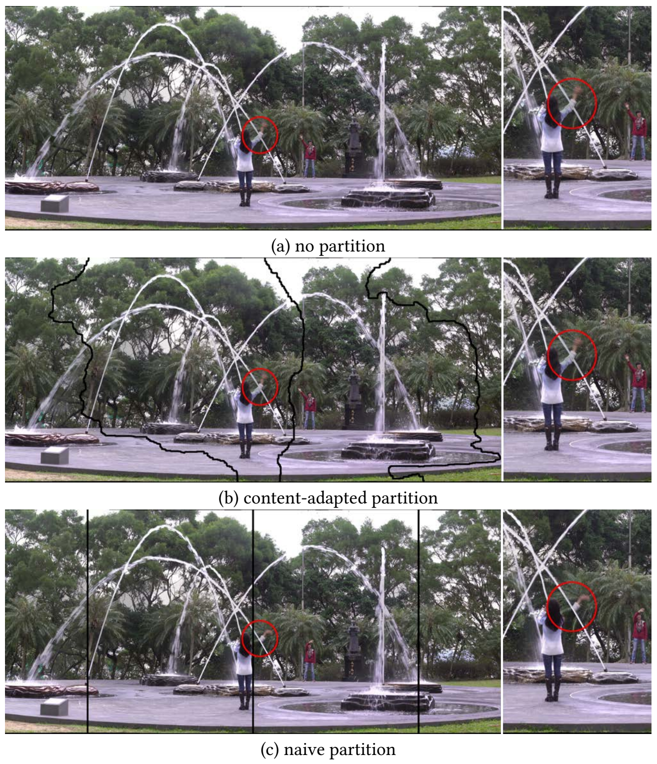
$$\max_i (A(P_i)/\bar{A}) \le 1 + \epsilon. \tag{21}$$

The positive constant ϵ sets a tolerance on the size difference compared to the target size $\bar{A} = \sum_{i=1}^{k} A(P_i)/k$. Empirically we let ϵ be 25% and set the number k of regions to the smallest value such that the largest part $\bar{A} \times (1 + \epsilon)$ still satisfies the memory limits.

We leverage a multi-constraint mesh partitioning method [Karypis 2003] to obtain the optimized domain partition (Fig. 10e). Since the optimal partition tries to avoid dynamically loopable pixels, the order of regions to be optimized has a very small influence on the final result. Once partitioning is complete, we optimize regions in random order. When optimizing each region, boundary pixels of neighboring regions that have already been processed serve as boundary constraints. We find that the difference in results by using different orders is small. Fig. 11 demonstrates the benefit of using this optimized partition. Thus, it is feasible to parallelize regions without common boundaries.

6 LOOP ASSEMBLY

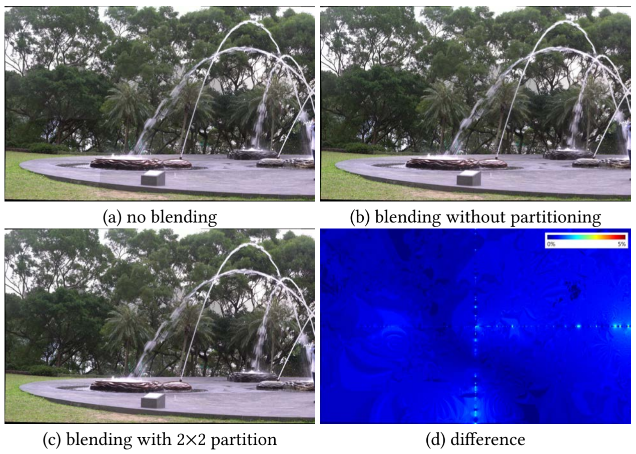
Having solved for the video source and looping parameters for each output pixel, we can begin to assemble the looping content into a panorama. Given a fixed output length, the output loop can adjust each looping period to the nearest integer number of loop instances [Liao et al. 2015]. We first upsample the optimized looping periods and start frames by the temporal scaling factor, and assemble an initial loop from the 4× spatially downsampled input videos of original length. Next, as in the previous paper, it is important to apply


(a) no partition

(b) content-adapted partition

(c) naive partition

Fig. 11. Out-of-core loop optimization achieves better results when the partition adapts to the content of the input video.


(a) no blending (b) blending without partitioning

(c) blending with 2×2 partition (d) difference

Fig. 12. Comparison of screened Poisson blending evaluated in-memory and out-of-core.

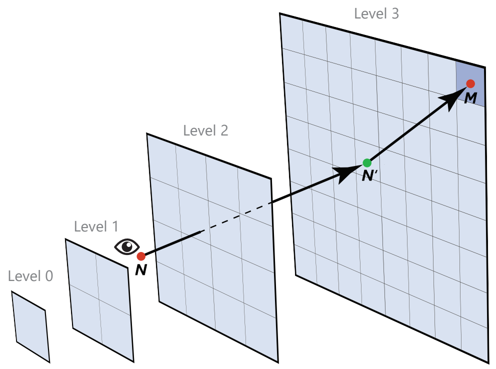


Fig. 13. Hierarchical tiling structure and priority computation. The arrows depict the distance used for priority computation in level-of-detail (z) and spatially (x, y) between the viewport centered at N and a tile M.

gradient-domain blending to further remove spatial seams and temporal pops. For the initial spatially downsampled loop, we solve a multigrid 3D screened Poisson problem to diffuse its spatiotemporal gradient inconsistencies. The resulting correction is upsampled by bilinear interpolation and added to the initial high-resolution loop. In the vicinity of inter-video spatial seams, a conjugate gradient solver is used to reduce blocky artifacts caused by upsampling. This significantly improves results near video segment boundaries, as shown in the right column of Fig. 5.

Out-of-core gradient blending Even though the Poisson blending is evaluated at the coarser level, memory may still become a bottleneck in the gigapixel scenario. Once again, we partition the spatial domain of the panorama, but in this case the position of boundaries is unimportant. We uniformly partition the panorama into rectangular regions that fit in memory, and perform Poisson blending over the regions in raster order. Similar to Summa et al. [2011], the results of previously solved neighboring regions serve as Dirichlet boundary condition for the subsequent regions. To validate the effectiveness of the partitioning, Fig. 12 compares the Poisson blending results with and without the use of domain partition for an example that can be solved entirely in memory. The differences are small, with no visible artifacts.

7 RENDERING

We design a renderer for users to interactively navigate through our gigapixel videos. The renderer seamlessly replays the region of the looping gigapixel video that overlaps with the current viewport. We dice the gigapixel video into a set of square video tiles in a 2D grid structure at multiple levels of detail (six in our results). As in image mipmapping, we use trilinear interpolation to render the current view. The approach is akin to multiresolution image viewers (e.g., Kopf et al. [2007]), but with looping video tiles instead of image tiles. The user can interactively navigate and zoom using the mouse.

Tile preloading strategy As with large image viewers, the video tiles are stored on disk and loaded (possibly over a network) into memory based on a priority schedule.

To make the best use of computing resources, we asynchronously load/discard tiles during navigation, always maintaining the ones with highest priority in memory. As a fallback case, we preload and use static images if the required video tiles are not loaded and available.

For a given current viewpoint N_{xy} and mipmap level N_z, the priority of the video tile located at M_{xy} and in mipmap level M_z is inversely proportional to its spatial distance and LOD distance to

N, and given by:

$$P(M, N) = \frac{1}{|M_z - N'_z|_1 + \lambda |M_{xy} - N'_{xy}|_1}, \quad (22)$$

where N' is the projection of N onto the mipmap level M_z. Note that in our mipmap system, we set the distance between adjacent tiles within the same level to 1 and set the distance between mipmap levels i and $i+1$ to 2^i. The parameter $\lambda = 1.5$, which tunes the relative importance between spatial and level distances, is determined empirically. Refer to the black line in Fig. 13 for an illustration of the priority computation.

With typical user interactions, we easily achieve frame rates above the 25fps rate of our video tiles. Please refer to our accompanying video for a demonstration.

8 RESULTS

Examples Our system is robust to a variety of challenging input panoramic scenes. In this paper, we demonstrate a few representative examples, including three gigapixel video panoramas to present the scalability of our algorithm: *seafront* (Fig. 1), *cityscape* at night (Fig. 14(top)), and *park* (Fig. 14(bottom)); and four smaller but challenging panoramas to validate our combinatorial optimization: *fountain* (Fig. 15(top)), *rocks* with challenging water motions (Fig. 14(bottom)), *commencement* (Fig. 16(top)) with many dynamic objects, and *square* (Fig. 16(bottom)) with a large crowd of pedestrians. For each example figure, the top row shows the entire panorama along with close-ups of areas with interesting activity. The bottom row shows the color-coded source video selection, the looping period, and the start frame for each panorama pixel.

The three gigapixel panoramas (*seafront*, *cityscape* and *park*) are sequentially captured in long time periods, which results in obvious inconsistent lighting conditions across the video grids, especially for *park* where the weather changes dramatically. This introduces a significant challenge to harmonize these videos. Our pre- and post-processing stages, our color correction, and Poisson gradient blending methods work well on achieving smooth and consistent transition. In *commencement* and *square*, the large number of individual dynamic objects make it extremely difficult to generate ideal loops. Our algorithm tries to remove those nonloopable objects while still preserving natural loops of more subtle motions. We also allow the user to locally edit the panorama to handle challenging regions (as described in Section 9). In *commencement*, the *Seam* operation is used in two regions to enforce spatial consistency. And in *square*, the *Freeze* operation is used in only one region to remove nonloopable motions. All types of editing operations are demonstrated in the accompanying video.

Performance Analysis All these examples use 8-second input video segments at 4K resolution and 25fps frame-rate. Their performance statistics are summarized in Table 1. Note that the input to each of the three gigapixel panoramas consists of over 180 video segments taken from approximately one hour of total captured video. To process the panoramas efficiently and without thrashing, we partition the gigapixel panorama domain into at most 20 regions. To further boost the performance, all our results are generated in parallel on a simple cluster with 4 nodes, each with 4-6 cores and 24-80GB RAM.

All gigapixel examples can be captured and processed in about 20 hours, while all other smaller examples take no more than 6 hours, as reported in Table 1. Please note that a considerable amount of time is spent on optimization, especially for the gigapixel panoramas due to expensive graph cut solver. Most of the stages can be naturally fully parallelized in clusters, and our out-of-core strategy manages to parallelize nonadjacent parts and decrease the processing time by approximately half.

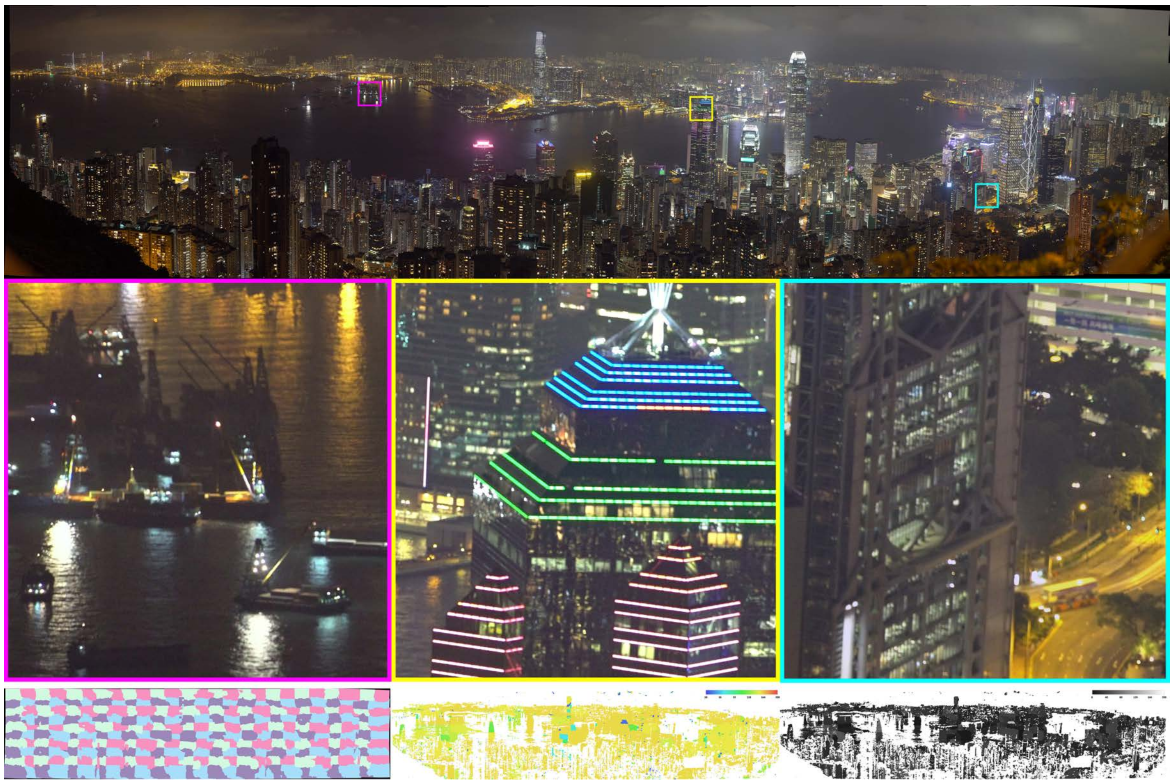
As for memory consumption, we notice that maximum in-use memory peaks at approximately 20GB. In particular, loop optimization is the most memory-intensive stage. For instance, the *seafront* panorama of 1197M pixels is partitioned into 20 regions with 60M pixels per region on average. The imbalance between the largest and smallest region is no more than 25%. On average, the graph cut processing on each region requires 14GB to store the source input video, and an additional 2.5GB for the optimization solver itself. Thus the source video data (used to evaluate the objective functional) dominates memory usage, and its size depends linearly on the region extent. Thanks to this linearity, our out-of-core processing strategy lets us adaptively determine the region size and thus preciously limit the memory requirements.

Rendering The results are best explored using the gigapixel renderer. The diced output tiles are 200 frames long and have a spatial resolution of 1024×1024. The accompanying video shows real-time navigation sessions of these panoramas at the 25fps native framerate of the input video data as well as closeup comparisons of the input videos and the resulting panoramas.

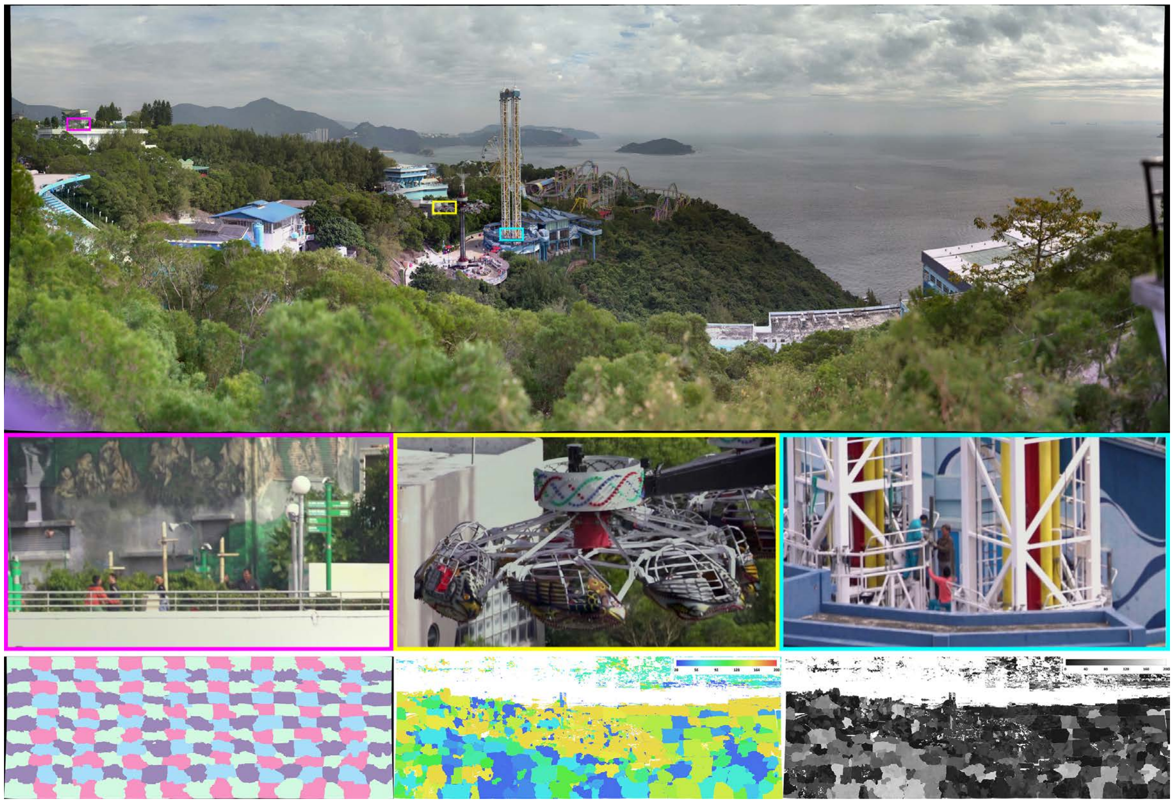
9 EXTENSION: SPATIALLY-SELECTIVE EDITING

Our system is able to automatically produce spatiotemporally consistent video loops. However, there may still exist regions where users would like to modify or adjust. One problem is that our optimization inherits the limitation of Liao et al. [2013] that the semantic relationship between objects might be inaccurate. When the foreground object has similar color and motion as the background, it might be difficult to preserve spatial continuity. For instance, a person can be split in the middle (Fig. 17(top)a). Another problem is that some nonloopable motions may be kept due to our dynamism preservation constraints. Imagine a scene with a person walking through. Our loop optimization may freeze the person, remove the person altogether, or even produce an unnatural loop where this person appears, walks along, and then disappears (Fig. 17(bottom)a). For these challenging cases, the best outcome that can be obtained is highly subjective, depending on whether the user is willing to sacrifice the continuous looping behavior in favor of a more dynamic scene.

We adapt our optimization framework to support interactive local adjustment over spatial consistency and dynamism on the gigapixel result. Our system provides an interface for the user to intuitively select a local region over the automatic result L_p and perform quick local update with new parameters to get L_s. To smoothly blend the local region of L_s onto L_p, we find an optimal seam between L_p and L_s and perform Poisson gradient blending across it. Given the user's selection mask at one key frame (Fig. 17b), we apply a robust object tracking method [Henriques et al. 2015] to continuously track the mask across the entire sequence and generate a conservative



(a) Source video id (b) Looping period (c) Start frame



(a) Source video id (b) Looping period (c) Start frame

Fig. 14. Visualization of the *cityscape* (top), and *park* (bottom) panoramas.

Table 1. Data and processing statistics for our looping panoramas.

panorama information						processing times (minutes)				
name	input segments	regions	viewer tiles	# edited regions	resolution	capture	preproc[†]	align[†]	optimize[†]	assemble & dice[†]
seafront	12×20	20	1559	0	1197MP	62	67	63	774	301
cityscape	10×26	20	1414	0	1051MP	68	71	52	661	287
park	11×17	15	1148	0	840MP	47	51	43	542	241
fountain	3×6	4[*]	116	0	82MP	6	4	6	40	15
commencement	5×7	4	138	2	94MP	9	8	10	65	29
rocks	5×14	5	311	0	219MP	17	16	15	195	67
square	8×8	5	383	1	259MP	15	15	14	242	70

[*] The fountain panorama could have been processed in memory. It is only partitioned to allow for parallelization.
[†] Parallelized over four computers.

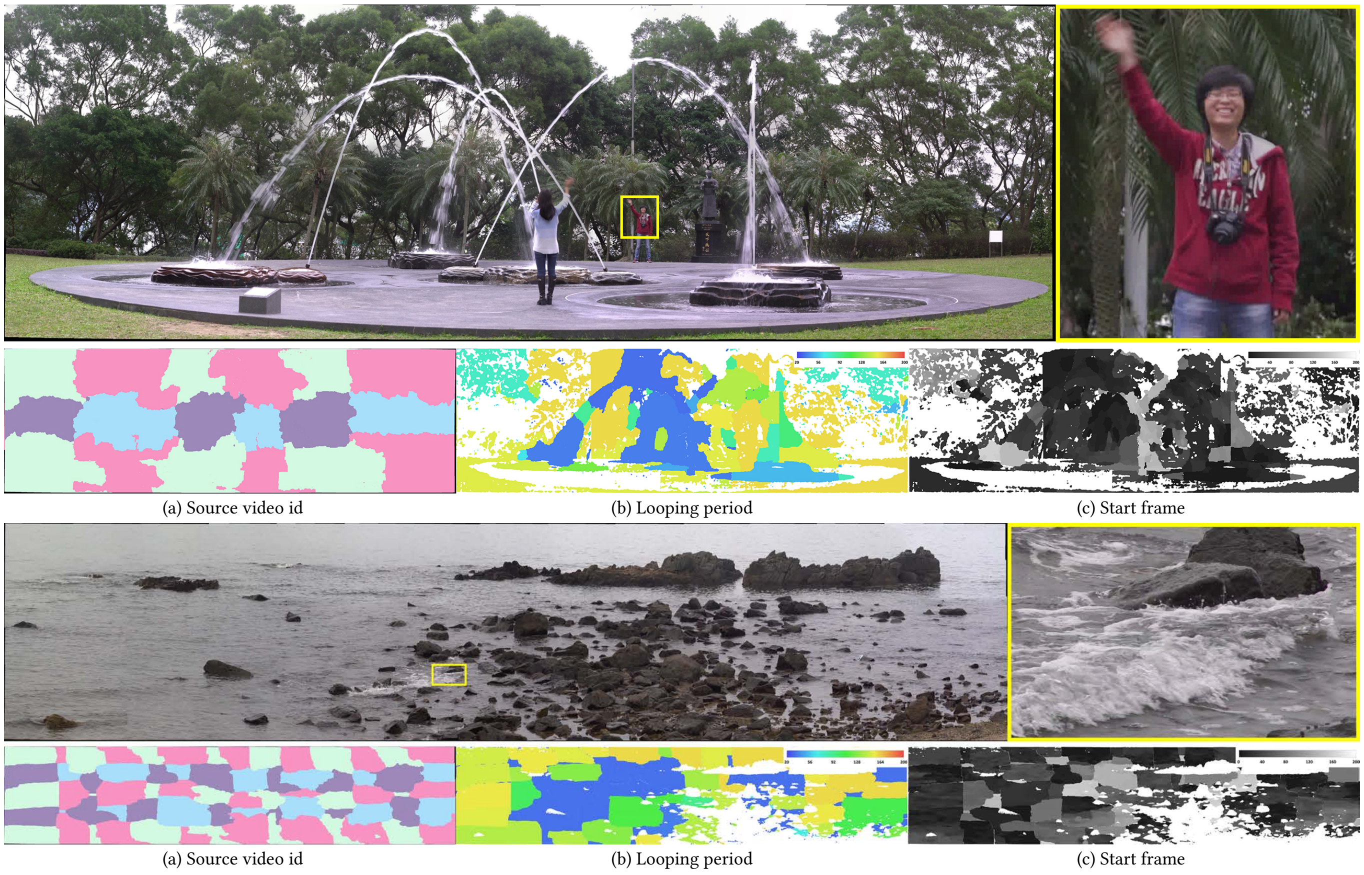
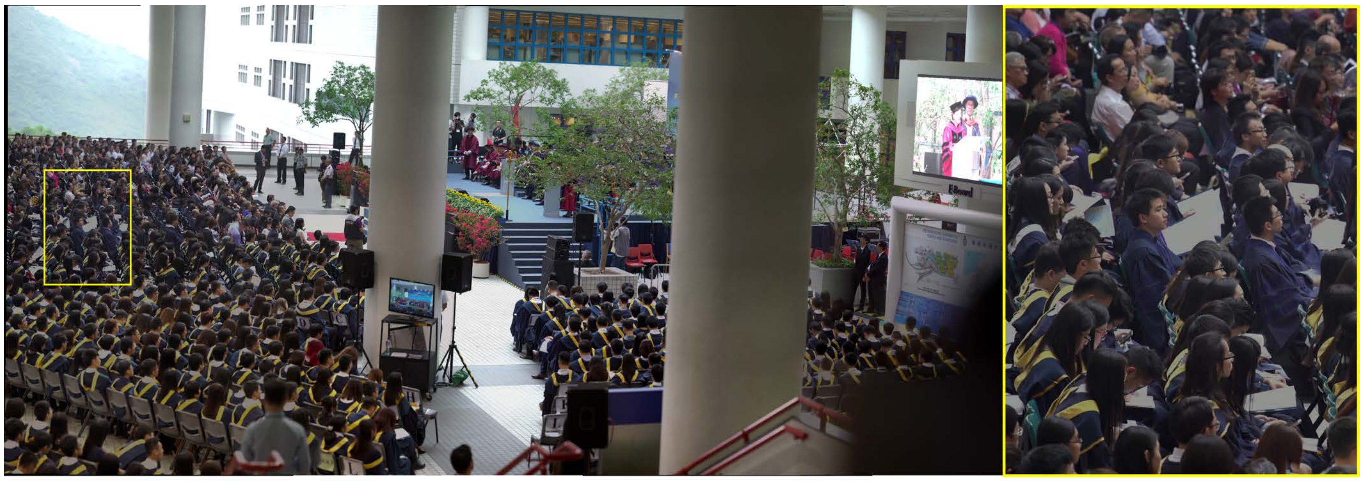

(a) Source video id (b) Looping period (c) Start frame
(a) Source video id (b) Looping period (c) Start frame

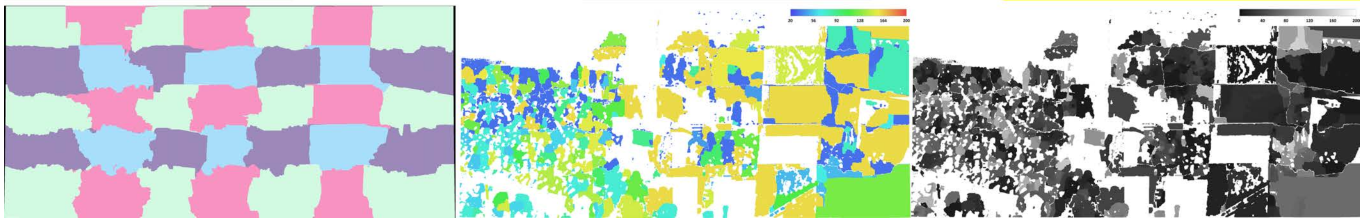
Fig. 15. Visualization of the *fountain* (top) and *rocks* (bottom) panoramas.

rectangle with the union mask M (Fig. 17c) bounding the target object. The video content inside the rectangle is then re-optimized according to user's choice among the three possible operations: *Seam*, *Freeze* and *Erase*. *Seam* is used to repair a broken object by locally increasing the parameter β in Eq. (8). *Freeze* can remove the object's motion within the selected region and set $p_x = 1$. *Erase* is implemented by decreasing β_{dynamic} to 0 in Eq. (8) to remove nonloopable objects. Fig. 17 shows the effect of these operations. To blend L_s into L_p, we first generate a conservative difference map D (Fig. 17d) inside this local region, with each pixel value $D(x)$ set to the largest color difference across all frames. This captures the entire motion of the object. Formally,

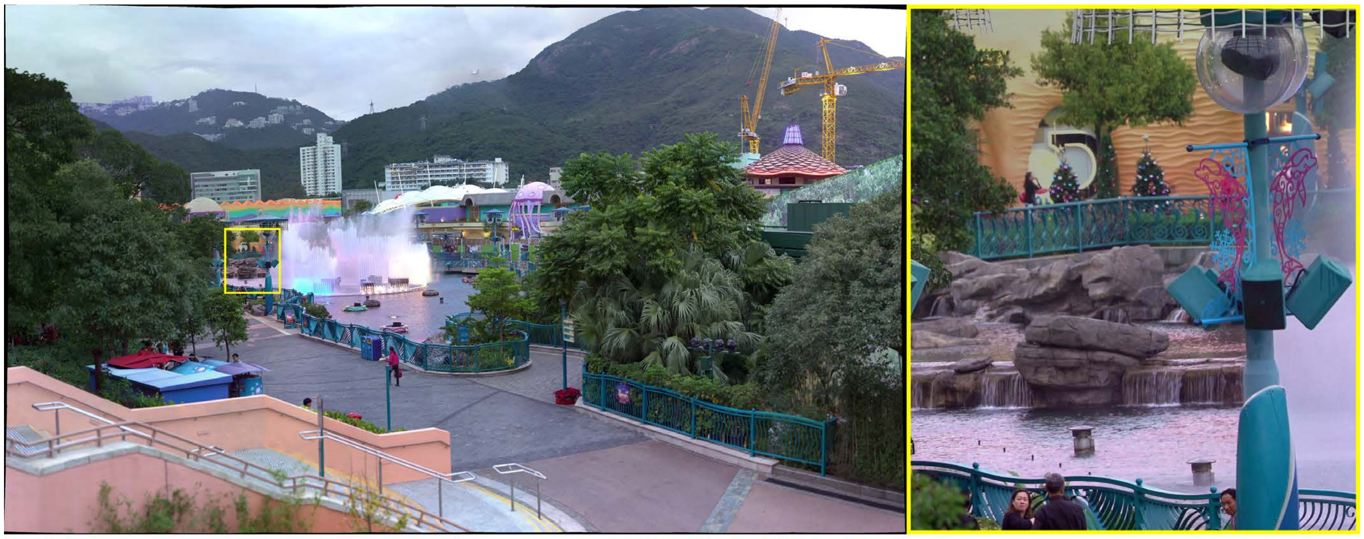
$$D(x) = \max_{0 \le t < T} \|L_p(x,t) - L_s(x,t)\| . \tag{23}$$

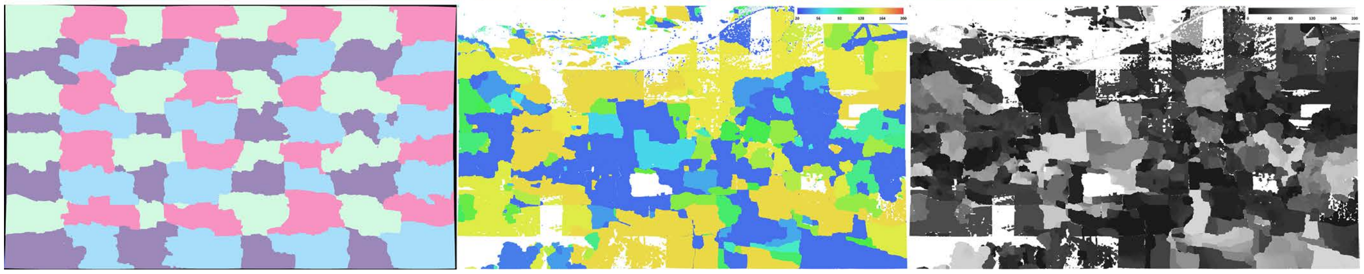
With this difference map D and the union object mask M, we apply graph cut to generate the blending seam (Fig. 17e), which is then used by Poisson gradient blending to get the final result (Fig. 17f). Once user editing is finalized, all the mipmap tiles affected by this modification are then updated. Please refer to the accompanying video for a demonstration.





(a) Source video id (b) Looping period (c) Start frame





(a) Source video id (b) Looping period (c) Start frame

Fig. 16. Visualization of the *commencement* (top) and *square* (bottom) panoramas after editing.

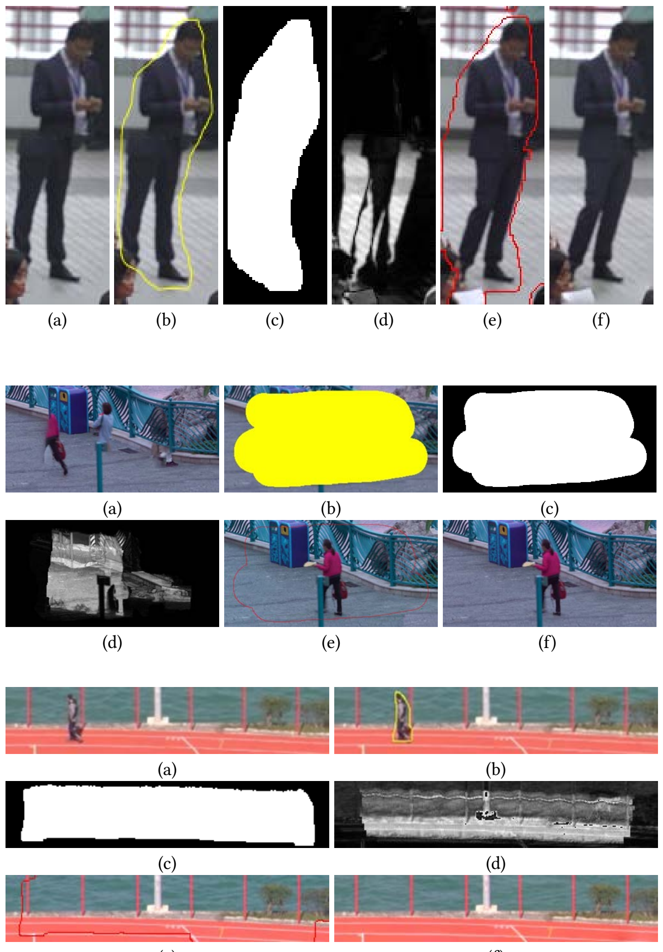


Fig. 17. Local editing examples, *Seam* (top), *Freeze* (middle) and *Erase* (bottom) with (a) for L_p, (b) for user's stroke, (c) for M, (d) for D, (e) for initial result and (f) for final result after Poisson gradient blending

10 DISCUSSION AND FUTURE WORK

We demonstrate a novel method to create high-resolution looping panoramic videos. The approach is to acquire a grid of spatially overlapping short videos and combine these into a single spatiotemporally consistent loop. Our pipeline includes an improved single-video looping optimization framework, a multi-input generalization for handling overlapping videos over a gigapixel domain with several acceleration strategies, and several pre- and post-processing steps required to achieve seamless results. The key contributing factor enabling the successful results is the formulation of a common combinatorial optimization for both stitching and looping.

While the technique works well for nature scenes which exhibit looping behavior, it cannot produce satisfactory results in some cases. Firstly, it may produce undesirable effects (e.g. a person disappearing behind a tree without re-appearing) in regions where there are nonloopable objects. To help mitigate this problem, we develop an interface for the user to perform local region editing with

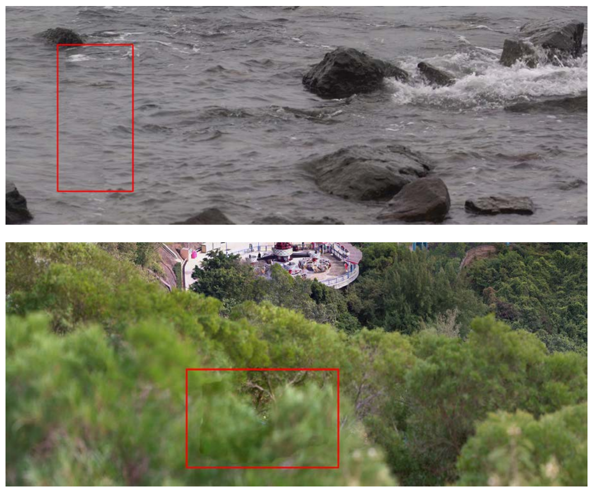


Fig. 18. Visible seam (highlighted in red) caused by waves(top) and leaves(bottom) which are relatively large relative to the segment sizes of this closeup panorama.

various operations. In future work, an automatic solution to detect and correct such regions without user assistance would be ideal. Secondly, visual discontinuities may arise when motions vary greatly at the overlapping regions (Fig. 18). A better stitching method may help improve spatial consistency in these challenging cases. Thirdly, the current system is constrained to similar depth-of-field across all input videos. Exploring ways to perform focus bracketing and merging the results during a preprocessing step could further improve the focus range of our resulting video panoramas. Finally, the current system is designed for input videos in a 2D grid layout, i.e, graph-coloring acceleration can only handle video segments with less than 50% mutual overlap. This could be extended to support inputs with more general layouts.

It would be interesting to explore the generalization to stereo panoramas as in [Couture et al. 2012]. The added challenge of maintaining temporal synchronization between the left and right views in each frame would link the choice of looping parameters for the two views. Existing methods for annotating static panoramas [Luan et al. 2008] or identifying their salient areas automatically [Ip and Varshney 2011] might be extended to operate on looping panoramas.

ACKNOWLEDGMENTS

The authors would like to thank Diego Nehab for discussions and suggestions in the beginning of the project, and Jinglu Wang, Songfang Han, Evan Han and Ji Cheng for their help in video data capture. This work was partly supported by Hong Kong GRF grants #619509 and #618513.

REFERENCES

A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, 23(3), 2004.
Aseem Agarwala. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph.*, 26(3):94, 2007.
Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic video textures. *ACM Trans.*

Graph., 24(3), July 2005.
Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Selectively de-animating video. *ACM Trans. Graph.*, 31(4), 2012.
Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Automatic cinemagraph portraits. *Computer Graphics Forum*, 32(4):17–25, 2013.
Jamie Beck and Kevin Burg. Cinemagraphs. http://cinemagraphs.com/, 2012.
Jean-Yves Bouguet. Pyramidal implementation of the affine Lucas-Kanade feature tracker: Description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.
Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *IJCV*, 74(1):59–73, 2007.
V. Couture, M. Langer, and S. Roy. Panoramic stereo video textures. *ICCV*, pages 1251–1258, 2011.
Vincent Couture, Michael S Langer, and Sébastien Roy. Perception of blending in stereo motion panoramas. *ACM Trans. on Applied Perception*, 9(3):15, 2012.
Dan B Goldman. Vignette and exposure calibration and compensation. volume 32, pages 2276–2288. IEEE, 2010.
João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.
Chris Hermans, Cedric Vanaken, Tom Mertens, Frank Van Reeth, and Philippe Bekaert. Augmented panoramic video. In *Computer Graphics Forum*, volume 27, 2008.
Cheuk Yiu Ip and Amitabh Varshney. Saliency-assisted navigation of very large landscape images. *IEEE TVCG*, 17(12), 2011.
Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. Cliplets: Juxtaposing still and dynamic imagery. *Proc. of UIST*, 2012.
George Karypis. Multi-constraint mesh partitioning for contact/impact computations. In *Proc. of ACM/IEEE conference on Supercomputing*, page 56. ACM, 2003.
M. Kazhdan and H. Hoppe. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph.*, 27(3), 2008.
Michael Kazhdan, Dinoj Surendran, and Hugues Hoppe. Distributed gradient-domain processing of planar and spherical images. *ACM Trans. Graph.*, 29(2), 2010.
V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. on Pattern Anal. Mach. Intell.*, 26(2), 2004.
Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F Cohen. Capturing and viewing gigapixel images. *ACM Trans. Graph.*, 26(3):93, 2007.
Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. In *ACM Trans. Graph.*, volume 22, pages 277–286, 2003.
Jing Liao, Neel Joshi, and Hugues Hoppe. Automated video looping with progressive dynamism. *ACM Trans. Graph.*, 32(4), 2013.
Jing Liao, Mark Finch, and Hugues Hoppe. Fast computation of seamless video loops. *ACM Trans. Graph.*, 34(6):197, 2015.
Jiangyu Liu and Jian Sun. Parallel graph-cuts by adaptive bottom-up merging. In *Proc. CVPR*, pages 2181–2188. IEEE, 2010.
Qing Luan, Steven M Drucker, Johannes Kopf, Ying-Qing Xu, and Michael F Cohen. Annotating gigapixel images. In *Proc. ACM UIST*, pages 33–36, 2008.
Microsoft Research. Image composite editor. http://research.\discretionary{-}{}{}microsoft.com/en-us/um/redmond/groups/ivm/ICE/, 2016.
Federico Perazzi, Alexander Sorkine-Hornung, Henning Zimmer, Peter Kaufmann, Oliver Wang, S. Watson, and Markus H. Gross. Panoramic video from unstructured camera arrays. *Computer Graphics Forum*, 34(2):57–68, 2015.
Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3), 2003.
Sujin Philip, Brian Summa, Valerio Pascucci, and P-T Bremer. Hybrid CPU-GPU solver for gradient domain processing of massive images. In *Intl. Conf. on Parallel and Distributed Systems*, 2011.
Sujin Philip, Brian Summa, Julien Tierny, Peer-Timo Bremer, and Valerio Pascucci. Distributed seams for gigapixel panoramas. *IEEE TVCG*, 21(3):350–362, 2015.
Sören Pirk, Michael F Cohen, Oliver Deussen, Matt Uyttendaele, and Johannes Kopf. Video enhanced gigapixel panoramas. *SIGGRAPH Asia 2012 Technical Briefs*, 2012.
Alex Rav-Acha, Yael Pritch, Dani Lischinski, and Shmuel Peleg. Dynamosaicing: Mosaicing of dynamic scenes. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 29(10), 2007.
Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proc. SIGGRAPH*, pages 489–498, 2000.
Laura Sevilla-Lara, Jonas Wulff, Kalyan Sunkavalli, and Eli Shechtman. Smooth loops from unconstrained video. *Computer Graphics Forum*, 34(4):99–107, 2015.
Petter Strandmark and Fredrik Kahl. Parallel and distributed graph cuts by dual decomposition. In *Proc. CVPR*, pages 2085–2092. IEEE, 2010.
Brian Summa, Giorgio Scorzelli, Ming Jiang, Peer-Timo Bremer, and Valerio Pascucci. Interactive editing of massive imagery made simple: Turning Atlanta into Atlantis. *ACM Transactions on Graphics (TOG)*, 30(2):7, 2011.
Brian Summa, Julien Tierny, and Valerio Pascucci. Panorama weaving: Fast and flexible seam processing. *ACM Trans. Graph.*, 31(4), 2012.
Richard Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1), 2006.
Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proc. ACM SIGGRAPH*, 1997.
James Tompkin, Fabrizio Pece, Kartic Subr, and Jan Kautz. Towards moment images: Automatic cinemagraphs. In *Proc. of the 8th European Conference on Visual Media Production (CVMP 2011)*, November 2011.
James Tompkin, Fabrizio Pece, Rajvi Shah, Shahram Izadi, Jan Kautz, and Christian Theobalt. Video collections in panoramic contexts. In *Proc. ACM UIST*, 2013.

Received MM YYYY; revised MM YYYY; final version MM YYYYY; accepted MM YYYY