Fast Computation of Seamless Video Loops

Jing Liao
Hong Kong UST

Mark Finch
Microsoft Research

Hugues Hoppe
Microsoft Research

Abstract

Short looping videos concisely capture the dynamism of natural scenes. Creating seamless loops usually involves maximizing spatiotemporal consistency and applying Poisson blending. We take an end-to-end view of the problem and present new techniques that jointly improve loop quality while also significantly reducing processing time. A key idea is to relax the consistency constraints to anticipate the subsequent blending, thereby enabling looping of low-frequency content like moving clouds and changing illumination. We also analyze the input video to remove an undesired bias toward short loops. The quality gains are demonstrated visually and confirmed quantitatively using a new gradient-domain consistency metric. We improve system performance by classifying potentially loopable pixels, masking the 2D graph cut, pruning graph-cut labels based on dominant periods, and optimizing on a coarse grid while retaining finer detail. Together these techniques reduce computation times from tens of minutes to nearly real-time.

CR Categories: I.3.0 [Computer Graphics]: General.

Keywords: video textures, cinemagraphs, blend-aware consistency

1 Introduction

The spatial resolution of videos is approaching that of digital photographs (e.g., 8-megapixel videos vs. 16-megapixel photos on current phones). Video content is thus becoming more prevalent, and we expect that as storage and bandwidth continue to scale, videos will displace photos as default capture medium. This paper focuses on computing short video loops for periodic motions (e.g., swaying trees, rippling water) in nature scenes, as such loops help convey a greater sense of presence than still images. Our goal is to create video loops without user assistance, much like the automatic mode for shooting photos on consumer devices, and to do so far more efficiently than prior methods.

Several techniques create looping videos from short input videos [e.g., Schödl et al. 2000; Kwatra et al. 2003; Agarwala et al. 2005; Couture et al. 2011; Liao et al. 2013]. The general approach is to assemble content from the original video such that 3D spatiotemporal neighborhoods of the resulting video loop are consistent with those of the input video. Typically this is cast as a combinatorial optimization with an objective of color consistency.

In this paper we describe an end-to-end pipeline for generating video loops of greater quality and with significantly less computational effort than prior methods. We introduce several new techniques that jointly address these two challenges.

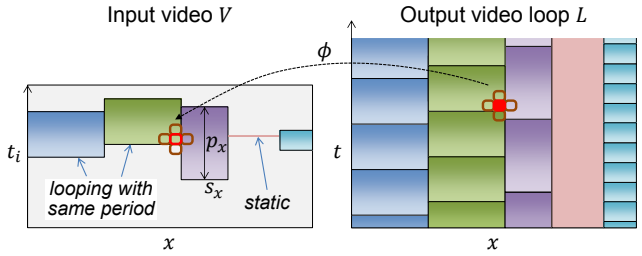


Figure 1: *A video loop L is formed from an input video V by repeating some temporal interval at each pixel x using a time-mapping function ϕ, specified using a period p_x and start frame s_x. The consistency objective is that for any output pixel color (shown in solid red), its spatiotemporal neighbors should have the same values as the corresponding neighbors in the input.*

Improved loop quality For many scenes, the color consistency constraints cannot be fully satisfied, resulting in spatial seams or temporal pops. Several approaches aim to reduce or hide these artifacts. Feathering or multiresolution splining is applied as a post-process [Schödl et al. 2000]. Gradient-domain Poisson blending improves results by diffusing spatiotemporal discrepancies [Agarwala et al. 2005]. The consistency constraints are adaptively attenuated by recognizing that discontinuities are less perceptible in high-frequency regions [Kwatra et al. 2003]. Troublesome scene regions are replaced by static (nonlooping) pixels, either using assisted segmentation [Agarwala et al. 2005; Tompkin et al. 2011; Bai et al. 2012; Joshi et al. 2012] or as part of the optimization [Liao et al. 2013]. However, for some scenes there may be little dynamism left.

Our key idea is to modulate the consistency objective in the loop synthesis algorithm to anticipate the subsequent step of Poisson blending and thereby provide greater flexibility for optimization. In particular, low-frequency image differences (e.g., smooth intensity changes due to illumination changes or moving clouds/smoke) can be ignored because they are easily corrected through Poisson blending. In contrast, distinct shape boundaries are not easily repaired. We show that the quality of video loops also benefits from giving a preference to longer periods.

Fast loop computation State-of-the-art video looping optimizations require many minutes of computation. We describe several algorithmic improvements that together reduce the processing time on a desktop PC to about 7 seconds for a 5-second Full HD video, i.e., nearly real-time.

We review prior work, including the framework of [Liao et al. 2013] upon which we build, then present our processing pipeline and contributions in Section 3.

2 Background and related work

Given an input video with color $V(x, t_i)$ at each 2D pixel x and input frame time t_i, the aim is to compute a video loop

$$L(x,t) = V(x, \phi(x,t)), \quad 0 \le t < T,$$

by determining a time-mapping function $\phi(x,t)$. Note that the loop content $L(x,\cdot)$ at a given position x is taken from the same pixel location $V(x,\cdot)$ in the input video (Figure 1).

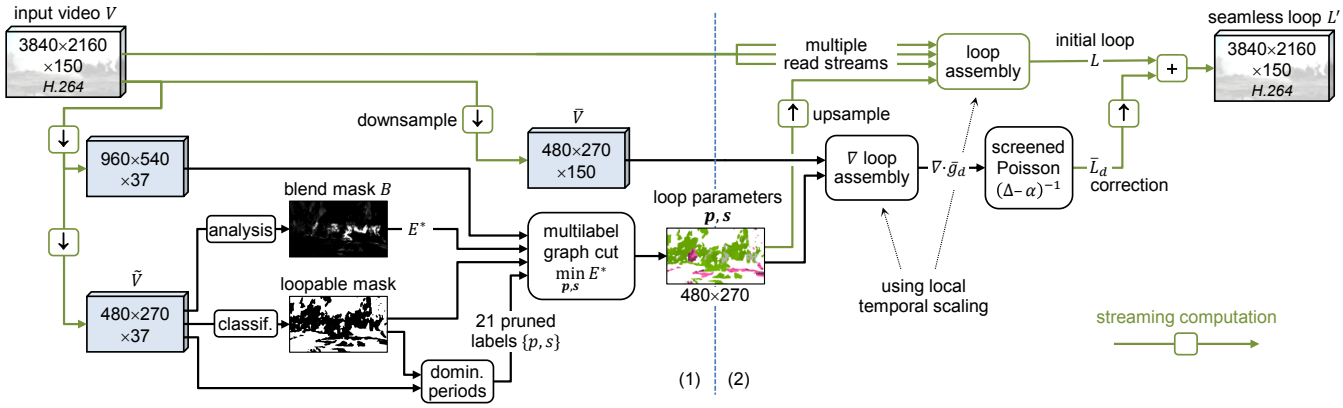


Figure 2: *Our processing pipeline has two stages: (1) determining looping parameters and (2) assembling a seamless Poisson-blended loop. Most computations operate on downsampled video. We obtain per-pixel periods and start frames by minimizing a consistency objective using a multilabel graph cut. We improve fidelity in this optimization by accessing detail from the next-finer resolution. We anticipate the subsequent Poisson blending by introducing a blend-aware objective E^* which spatially attenuates consistency based on a blend mask. Classification of loopable pixels in the input video reduces the dimension of the search space. Identification of dominant periods reduces the set of labels. Local temporal scaling lets us assemble an ordinary 5-second loop. Screened Poisson blending removes spatial seams and temporal pops. It involves gathering sparse gradient differences g_d in the downsampled loop, solving a multigrid Poisson problem, upsampling the resulting correction, and merging it with the initial high-resolution loop. We reduce memory usage by opening multiple read streams on the input video.*

Techniques can be contrasted by their definitions of the temporal mapping ϕ. Schödl et al. [2000] transition regions simultaneously by finding compatible frames. If $V(\cdot, t_A) \approx V(\cdot, t_A + p)$, a good mapping is $\phi(x,t) = t_A + (t \bmod p)$. Kwatra et al. [2003] allow pixels to transition at different times. They obtain a loop of period p by solving a binary 3D graph cut over variables $b(x,t) \in \{0,1\}$ such that $\phi(x,t) = t_0 + (t \bmod p) + b(x, (t \bmod p))\, p$. Agarwala et al. [2005] create a loop from a panoramic-sweep video by allowing the time-mapping function $\phi(x,t) = \delta(x, (t \bmod p))$ to have arbitrary temporal offsets δ into the stabilized input video and solving a multilabel 3D graph cut.

Several techniques exploit user guidance to create loops. The interactive tool of Tompkin et al. [2011] juxtaposes static and looping regions to create cinemagraphs [Beck and Burg 2012]. Joshi et al. [2012] develop a set of idioms (static, play, loop, and mirror loop) to combine several spatiotemporal segments from a source video, thus emphasizing particular scene elements or forming a narrative. Bai et al. [2012] apply spatial warping to the video content to selectively de-animate content. Guided by user strokes, their approach removes large-scale motions while preserving high-frequency movement. Bai et al. [2013] use tracking on handheld video to create portrait cinemagraphs. Sevilla-Lara et al. [2015] use morphing to create a loop for the case of a contiguous foreground object that can be segmented from its background.

Our work builds on the automated technique of Liao et al. [2013] which optimizes a looping period p_x and start frame s_x at each pixel:

$$\phi(x,t) = s_x + ((t - s_x) \bmod p_x).$$

In regions with nonloopable content, a pixel may be assigned the period $p_x = 1$, which makes it static by freezing its 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

$$E(\mathbf{p}, \mathbf{s}) = E_{\text{consistency}}(\mathbf{p}, \mathbf{s}) + E_{\text{static}}(\mathbf{p}, \mathbf{s}). \tag{1}$$

The term $E_{\text{consistency}} = E_{\text{spatial}} + E_{\text{temporal}}$ measures the spatiotemporal consistency of neighboring colors in the video loop with respect to the input video [Agarwala et al. 2005]. The spatial term sums consistency over all spatially adjacent pixels x, z:

$$E_{\text{spatial}} = \sum_{\|x-z\|=1} \Psi_{\text{spatial}}(x,z)\, \gamma_s(x,z) \quad \text{with}$$

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

and the temporal term sums consistency across the loop end-frames s_x and $s_x + p_x$ at each pixel:

$$E_{\text{temporal}} = \sum_x \begin{pmatrix} \|V(x,s_x) - V(x,s_x+p_x)\|^2 + \\ \|V(x,s_x-1) - V(x,s_x+p_x-1)\|^2 \end{pmatrix} \gamma_t(x).$$

The factors γ_s, γ_t attenuate consistency in high-frequency regions [Kwatra et al. 2003; Bai et al. 2012]. Finally, the term E_{static} assigns a penalty to static pixels to prevent a trivial all-static solution.

The minimization 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 is found using a multilabel graph cut algorithm, which iterates through the set of labels several times [Kolmogorov and Zabih 2004]. For each label α, it solves a 2D binary graph cut to determine if the pixel should keep its current label or be assigned label α — this is referred to as alpha expansion. Lastly, spatiotemporal feathering is applied to help mask seams in the resulting video loop.

3 Overview and contributions

Figure 2 summarizes our processing pipeline. Like Liao et al. [2013], we optimize per-pixel periods $\mathbf{p}$ and start frames $\mathbf{s}$. And like Agarwala et al. [2005], we diffuse inconsistencies using gradient-domain (Poisson) blending. Our contributions include:

- Coarsening the 2D optimization domain while maintaining the accuracy of finer-scale detail.
- Modifying spatiotemporal consistency using a blend mask to anticipate the opportunity provided by Poisson blending.
- Classifying loopable pixels to reduce the optimization domain using a 2D binary mask.

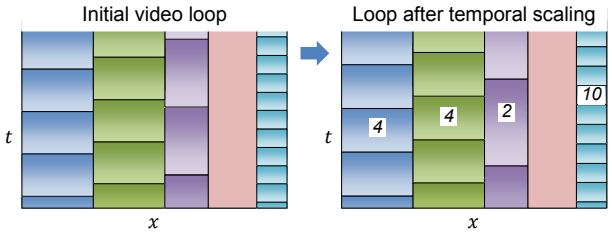


Figure 3: *Local temporal scaling creates an ordinary video loop by temporally stretching or shrinking each looping period to an integer number of loop instances.*

- Identifying the dominant periods of loopable pixels to prune the candidate loop periods and start frames.
- Removing an undesired bias towards shorter loops.
- Using a screened Poisson formulation to enable a fast multigrid algorithm for the gradient-domain blending.
- Applying local temporal scaling to allow creation of an ordinary short loop even with the flexibility of differing per-pixel periods.
- Assembling the loop as a streaming computation using multiple read streams to reduce memory.

We next present these techniques in greater detail.

4 Local temporal scaling

The optimization over $\mathbf{p}, \mathbf{s}$ allows pixels to have different looping periods. One drawback is that such a representation is not supported in common video playback components. While it is possible to create a repeating loop whose length is the least common multiple of all periods in $\mathbf{p}$, such a loop is often impractically long.

Our approach is to temporally scale the content such that each looping period adjusts to the nearest integer number of instances in a fixed-size loop, e.g., 5 seconds long (see Figure 3). For example, when generating a 150-frame loop, given a pixel whose looping content is 40 frames long, we temporally scale the content by a factor of 0.9375 to obtain exactly 4 loop instances. Mathematically, we obtain a perturbed time-mapping function $\phi'(x,t)$, in which incrementing the output time t sometimes repeats or skips an input video frame.

The temporal scaling does not introduce appreciable artifacts because all pixels with the same period are adjusted by the same factor, so that their spatiotemporal relationships are preserved. It is only at the boundaries between pixels with different periods that spatial seams could worsen. Fortunately, the spatial consistency cost $\Psi_{\text{spatial}}(x,z)$ between two neighboring pixels with different periods already makes the worst-case assumption that these periods are independent (i.e., relatively prime), so generally these boundaries lie along pixels with relatively unchanging content and thus temporal scaling has little visible effect.

The approach is related to the *independent looping regions* created by Liao et al. [2013], which also involve pixels with a shared period. Their goal is to selectively freeze these regions to spatially control dynamism. Freezing a region can be viewed as an extreme case of temporal scaling.

For all results in this paper, we use this local temporal scaling scheme to generate ordinary 5-second loops (with $T=150$ frames). As shown in Figure 2, given the loop parameters $\mathbf{p}, \mathbf{s}$ computed in the multilabel graph cut, local temporal scaling is used both to define an initial loop L and to formulate the Poisson blending which determines the final loop L'.

No correction | Spatiotemp. feathering | Poisson blending

Figure 4: *Poisson blending more effectively diffuses errors than feathering, as visualized here near spatial seams. Temporal pops are similarly attenuated as seen in the supplemental video.*

5 Improved looping quality

In this section we describe several techniques to obtain better loops, i.e., with greater dynamism and improved spatiotemporal consistency.

5.1 Screened Poisson blending

Like Agarwala et al. [2005], we apply Poisson blending [Pérez et al. 2003] to video looping, so that discrepancies at the stitching boundaries are diffused over the full domain, unlike with a finite feathering filter (Figure 4). Whereas they use Dirichlet constraints at the boundaries of user-specified looping regions, we introduce a weak prior based on the colors in the initial (unblended) loop L and optimize the blended colors L' over the full 3D domain. In our notation, we seek

$$\min_{L'} \left(E'_{\text{consistency}}(L') + \alpha \|L' - L\|^2 \right),$$

where $E'_{\text{consistency}} = E'_{\text{spatial}} + E'_{\text{temporal}}$ measures *gradient-domain* spatiotemporal consistency by comparing *differences* of adjacent colors in the final loop and the original video. The term E'_{spatial} uses

$$\Psi'_{\text{spatial}}(x,z) = \frac{1}{T} \sum_{t=0}^{T-1} \left(\begin{array}{c} \left\| \begin{array}{c} (L'(z,t) - L'(x,t)) - \\ (V(z,\phi'(x,t)) - V(x,\phi'(x,t))) \end{array} \right\|^2 + \\ \left\| \begin{array}{c} (L'(z,t) - L'(x,t)) - \\ (V(z,\phi'(z,t)) - V(x,\phi'(z,t))) \end{array} \right\|^2 \end{array} \right).$$

Similarly, E'_{temporal} uses

$$\frac{p_x}{T} \sum_{t=0}^{T-1} \left(\begin{array}{c} \left\| \begin{array}{c} (L'(x,t+1) - L'(x,t)) - \\ (V(x,\phi'(x,t)+1) - V(x,\phi'(x,t))) \end{array} \right\|^2 + \\ \left\| \begin{array}{c} (L'(x,t+1) - L'(x,t)) - \\ (V(x,\phi'(x,t+1)) - V(x,\phi'(x,t+1)-1)) \end{array} \right\|^2 \end{array} \right),$$

with wraparound temporal access to L'. Note that $E'_{\text{consistency}}$ reduces to $E_{\text{consistency}}$ when $L'=L$ (assuming that both are defined using the temporally scaled ϕ'). The minimization is equivalent to $\min_{L'} \|\nabla L' - g\|^2 + \alpha \|L' - L\|^2$ where g is a gradient field defined from V and ϕ'. Its solution corresponds to a screened Poisson equation [Bhat et al. 2008],

$$(\Delta - \alpha) L' = \nabla \cdot g - \alpha L.$$

The absence of irregular Dirichlet boundaries lets us solve the linear system using a simple multigrid algorithm. The algorithm coarsens the domain in both the spatial and temporal dimensions with a 3D box filter. Numerically precise solutions can be obtained using ten multigrid V-cycles and two iterations of Gauss-Seidel relaxation per level on each leg of a V-cycle. However, we find that using just three multigrid V-cycles yields solutions with 0.37% rms error for Full HD video, which is sufficiently precise for 8-bit color channels.

Figure 5: *Visualization of the blend mask B computed for two example input videos. Bright luminance corresponds to $B = 1$, i.e., regions that contain sharp transitions and are therefore not easily blended. The regions highlighted in red have low values of B, reflecting the fact that they are easily blended.*

5.2 Blend-aware consistency metric

A weakness of solving $\min_{\mathbf{p},\mathbf{s}} E(\mathbf{p},\mathbf{s})$ as in prior work is that it fails to account for the fact that Poisson blending may yet diffuse the inconsistencies to obtain a seamless solution, i.e., with lower $E'_{\text{consistency}}$. As a simple example, consider a scene whose illumination brightens slowly over time. Temporal consistency compares the colors near the loop start and end frames. Because the colors differ, the optimization is likely to favor short loops or may even freeze the scene altogether, whereas Poisson blending would smooth away the low-frequency illumination change even for a long loop. Although for this case one could globally adjust illumination as a preprocess, the benefit of Poisson blending is that it applies more generally. For instance, it is also effective spatially and on local discrepancies.

Ideally, we would like to minimize the gradient-domain-based

$$E'(\mathbf{p},\mathbf{s},L') = E'_{\text{consistency}}(\mathbf{p},\mathbf{s},L') + E_{\text{static}}(\mathbf{p},\mathbf{s}) \qquad (2)$$

over both the combinatorial loop parameters $\mathbf{p},\mathbf{s}$ and the final blended colors L'. However, this is challenging because changes in $\mathbf{p},\mathbf{s}$ result in structural changes to the desired gradient field g.

Instead, our approach is to minimize (1) but using a modified objective $E^* = E^*_{\text{consistency}} + E_{\text{static}}$ where the consistency metric is *blend-aware*.

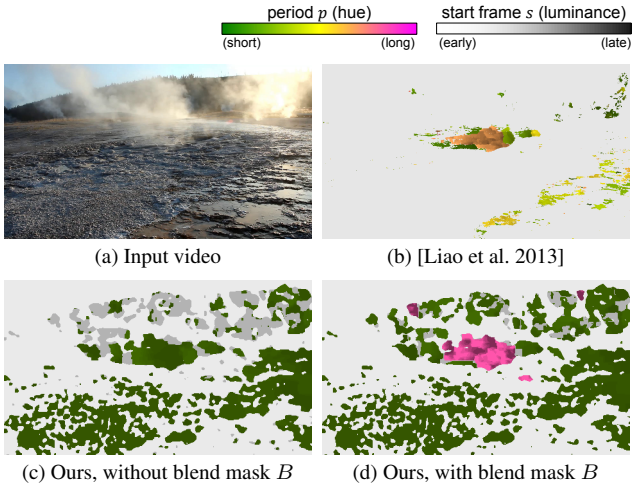
From the input video we compute a spatial *blend mask* B that is used to modulate the spatial and temporal consistency terms. Intuitively, $B(x)$ is small if pixel x is not traversed by any sharp boundary in the input video, i.e., if it is in a blendable region.

Conceptually, we want to compute the mask B at each pixel based on the maximum temporal derivative of the input video's highpass signal. As we shall see in the following derivation, this is well approximated simply by the maximum temporal derivative.

Let $V_L = V * G$ be a spatially blurred version of the input video, obtained with a spatial Gaussian filter G. The highpass signal is therefore $V_H = V - V_L$. Its temporal derivative is

$$\begin{aligned} V_H^{t+1} - V_H^t &= \left(V^{t+1} - V^{t+1} * G\right) - \left(V^t - V^t * G\right) \\ &= \left(V^{t+1} - V^t\right) - \left(V^{t+1} - V^t\right) * G \\ &\approx \left(V^{t+1} - V^t\right). \end{aligned}$$

The approximation exploits the fact that the temporal derivatives have lower magnitude and become negligible after the spatial blur.



(a) Input video (b) [Liao et al. 2013]

(c) Ours, without blend mask B (d) Ours, with blend mask B

Figure 6: *Comparison of the looping parameters computed in Liao et al. [2013] and in our method without/with the blend-aware consistency metric (hue indicates period, brightness indicates start frame, white pixels are nonloopable, and gray pixels are assigned static). Result (c) differs from (b) due to other improvements in Section 5. Blend-aware consistency enables more pixels to loop, and some pixels to have longer periods.*

Before Poisson blending After Poisson blending

Figure 7: *As shown in these close-ups, blend-aware consistency leads to seams in the initial loop L, but these are smoothed during the subsequent Poisson blending, resulting in a better overall result L'. The accompanying video shows similar behavior for the associated temporal discontinuities.*

Thus for each pixel position x, we assign the blend mask

$$B(x) = \text{clamp}\left(\max_t \left(V(x,t+1) - V(x,t)\right) \cdot c_b, 0, 1\right),$$

with the scaling factor $c_b = 1.5$. The resulting mask B is illustrated in Figure 5.

We use $B(x)$ to modulate the spatial and temporal consistency terms, as highlighted in blue:

$$E^*_{\text{spatial}} = \sum_{\|x-z\|=1} \Psi_{\text{spatial}}(x,z)\, \gamma_s(x,z)\, \max(B(x), B(z))$$

and

$$E^*_{\text{temporal}} = \sum_x \left(\begin{array}{c} \|V(x,s_x) - V(x,s_x+p_x)\|^2 + \\ \|V(x,s_x-1) - V(x,s_x+p_x-1)\|^2 \end{array} \right) \gamma_t(x)\, B(x).$$

Note that the new factors supplement the local edge-strength modulation factors $\gamma_s(x), \gamma_t(x)$ from prior work. It is worth emphasizing the differences. The edge-strength factors γ are based on *average* or *median* differences of pixel colors. They reflect the fact that seams are less perceptible in *high-frequency* (textured) regions and pops are less perceptible on highly dynamic pixels. In

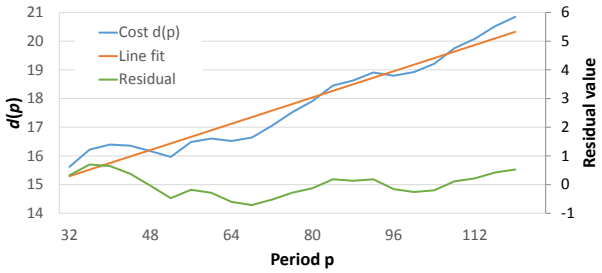


Figure 8: *Temporal cost $d(p)$ of the best synchronous loop as a function of the period p, together with a linear fit and its residual $d_R(p)$, for the palmtrees in the second row of Figure 9.*

contrast, the blend mask B is based on *maximum* differences, and reflects the fact that seams and pops are less perceptible away from moving *sharp* boundaries after gradient-domain blending.

The net effect is to focus consistency constraints on regions where color discrepancies are less likely to be corrected by subsequent blending. Figures 6 and 7 show an example.

Currently, the computation of B is conservative in that it considers the full input video even though the generated video loop accesses only a temporal subset. Future work could explore updating B somehow based on the content in the selected video loop.

5.3 Adapted temporal costs to promote longer loops

We find empirically that temporal consistency tends to favor shorter video loops. Liao et al. [2013] counter this by constraining all looping periods to be at least one second long. Many of their results use these minimal loops. There is a simple intuitive explanation. The difference between a given frame and progressively later frames tends to increase as small differences (lighting variations, shifting objects) gradually accumulate in the scene.

To analyze this, we define the difference $d^{0.8}(V(\cdot,t_1),V(\cdot,t_2))$ of two video frames t_1, t_2 as the 80$^{\text{th}}$-percentile absolute error of corresponding pixels. This percentile error is more robust than the traditional L^2 metric as it ignores the large errors in nonloopable regions, which are likely made static in any case. We speed up the evaluation of $d^{0.8}$ by sampling it on a 25% subset of image pixels.

For a *synchronous loop* in which all pixels share the same period p and start frame s, the cost as measured at the two nearest transition frames [Schödl et al. 2000] is

$$d(p,s) = \begin{pmatrix} d^{0.8}\left(\tilde{V}(\cdot,s),\tilde{V}(\cdot,s+p)\right) + \\ d^{0.8}\left(\tilde{V}(\cdot,s-1),\tilde{V}(\cdot,s+p-1)\right) \end{pmatrix}.$$

In Figure 8, we visualize $d(p) = \min_s d(p,s)$, the cost of the best synchronous loop for each loop period. We see that even for a scene with some natural cyclic motion (in this example, a period of about 17*4 frames), although $d(p)$ dips slightly as expected, the upward trend prevents this from becoming a minimum.

In general, we would prefer to use a longer loop if possible (i.e., if scene elements are loopable) because (1) it increases the variety of unique content in the resulting video and (2) it reduces the frequency of temporal blending artifacts.

To address this, for each input video we compute $d(p)$ and fit an affine model $\tilde{d}(p) = mp + b$ as shown by the red line in Figure 8. We redefine the temporal consistency cost at any pixel x to subtract this affine model:

$$E^{*(\text{adapted})}_{\text{temporal}}(x) = E^{*(\text{old})}_{\text{temporal}}(x) - \tilde{d}(p_x).$$

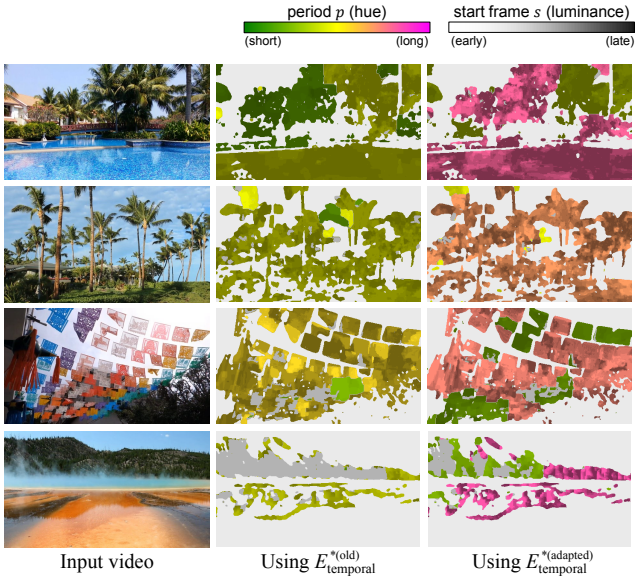


Figure 9: *The adapted temporal costs promote longer loop periods (seen as a shift in hue). The accompanying video shows the associated improvement in loop quality.*

Also, local minima in the residual costs $d_R(p,s) = d(p,s) - \tilde{d}(p)$ are used later in Section 6.4 to select good candidate periods for loop optimization. Some example results are shown in Figure 9.

We also explored encouraging greater dynamism by measuring the variance of the content within the loop chosen at each pixel. However, this tends to favor short loops with fast transient activity rather than a more natural animation.

6 Fast loop computation

We explore several acceleration techniques. To simplify the presentation, we assume that the input video and the computed loop are both 5-seconds long and sampled at 30 frames/sec.

6.1 Spatiotemporal downsampling

We first compute a spatiotemporally downsampled version $\tilde{V}$ of the input video using a 3D box filter. The temporal scaling factor is always 4. The spatial scaling factor is a power of two such that the resulting vertical size is no larger than 350. For example, an input video with resolution $3840\times2160\times150$ is scaled to $480\times270\times37$. All computations are performed on $\tilde{V}$, except the graph cut which defines its objective using the next-finer-level detail (Section 6.5) and Poisson blending which outputs at full resolution (Section 6.6).

6.2 Classification of loopable pixels

Given the downsampled video $\tilde{V}$, we quickly identify spatial regions that are unlikely to form good loops, so as to reduce the optimization effort to a smaller subset of pixels. The approach is to classify each pixel into one of 3 classes: *unchanging* (constant in $\tilde{V}$), *unloopable* (dynamic but unable to loop), or *loopable*. Pixels classified as unchanging or unloopable are made static and not considered in the optimization. Classification should be conservative, erring on the side of labeling a pixel as loopable, so that at worst, the optimization can still freeze the pixel.

As described next, we compute initial binary scores $(0,1)$ for each of the three classes at each pixel independently, spatially smooth the scores, and finally classify each pixel based on its maximum score.

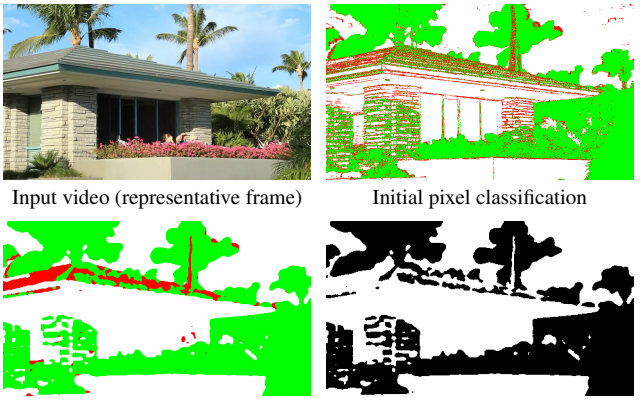


Figure 10: *Classification of 2D pixels into unchanging (white), unloopable (red), and loopable (green). Unchanging and unloopable pixels are made static, whereas loopable pixels define a mask (black) of the subset of pixels to be optimized during the graph cut.*

Initial scores We compute initial binary scores $(0, 1)$ for each of the three classes at each pixel as follows. Given position x and color channel $c \in \{0, 1, 2\}$, we define (using $\epsilon = \frac{10}{255}$):

$$\text{rises}(x, c) = \exists t_1, t_2 \text{ s.t. } t_1 < t_2 \wedge \tilde{V}_c(x, t_2) - \tilde{V}_c(x, t_1) > \epsilon$$
$$\text{falls}(x, c) = \exists t_1, t_2 \text{ s.t. } t_1 < t_2 \wedge \tilde{V}_c(x, t_1) - \tilde{V}_c(x, t_2) > \epsilon.$$

These predicates are computed in a single traversal of the video by tracking running minimum and maximum values at each pixel.

We then assign (where $\veebar$ denotes *xor*)

$$\text{label}(x) \leftarrow \begin{cases} \text{unchanging} & \text{if } \forall c \ \neg\text{rises}(x, c) \wedge \neg\text{falls}(x, c) \\ \text{unloopable} & \text{if } \exists c \ \text{rises}(x, c) \veebar \text{falls}(x, c) \\ \text{loopable} & \text{otherwise.} \end{cases}$$

Spatial smoothing We apply a Gaussian blur ($\sigma = 7$ pixels) to each of the three score fields. This serves to remove small islands of static pixels in larger looping regions as well as small islands of dynamic pixels in static regions, both of which are visually distracting.

Voting Finally, we classify each pixel according to its maximum smoothed score. Figure 10 shows the effect of smoothing on the final classification and the resulting binary mask of loopable pixels.

We could omit the computation of the blend mask B (Section 5.2) for nonloopable pixels, but this does not result in a speedup. On the other hand, we exclude nonloopable pixels in our estimates of dominant looping periods (later in Section 6.4) and find that this significantly improves quality.

6.3 Masked 2D graph cut

When nonloopable pixels are excluded from the graph cut, the graph is no longer a regular 2D grid. Although one could invoke a version of graph cut designed for general graphs, we find that it is much more efficient to preserve the regular connectivity of the 2D graph and instead modify the graph-cut solution to account for the binary mask. Specifically, we omit computing the data cost terms for any nonloopable pixel (since it cannot change). And, for any nonloopable pixel x adjacent to a loopable pixel z, we transfer the smoothness cost $E^*_{\text{spatial}}(x, z)$ to the data cost of the loopable pixel z. For parallelism, our implementation builds on the multithreaded graph cut approach of Liu and Sun [2010].

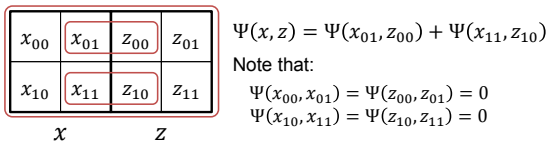


Figure 11: *The spatial consistency cost of two adjacent coarse-scale pixels x and z is the sum of the spatial consistency costs of their boundary-adjacent fine-scale pixels. Within the interior of each (2×2) block, spatial consistency is guaranteed because the fine-scale pixels share the same label (period and start frame).*

The fraction of nonloopable pixels varies significantly across videos, as indicated in Table 1. It results in a speedup of about a factor 1.6 overall (Table 4).

6.4 Pruned candidate labels

Liao et al. [2013] consider all periods $\{p\}$ and all possible start frames $\{s\}$ for each period. For a $4\times$ temporally downsampled 5-second input video (i.e., 37 frames), and a minimum loop period of 8 frames, this results in a multilabel graph cut with 502 labels $(s = 0, 1, \ldots 37 \,|\, p = 1; s = 0, 1, \ldots 29 \,|\, p = 8; s = 0, 1, \ldots 28 \,|\, p = 9; \ldots; s = 0 \,|\, p = 37)$. Performing alpha expansion over all these labels is costly. We heuristically prune this set to just 21 candidates as follows.

We find that it is useful to identify two dominant periods in the input video, a long period to provide greater content variety, and a shorter one as a fallback for regions on which there are no good long loops.

We use the adjusted synchronous temporal costs $d_R(p, s)$ from Section 5.3 to identify the most promising synchronous loop $(p_1, s_1) = \arg\min_{(p,s)} d_R(p, s)$. We compute these costs only over the loopable pixels identified in Section 6.2. Also, we disallow loop periods greater than 4 seconds ($p > 30$) because loop lengths that approach the length of the input video have insufficient variety of start frames to allow good loop creation. We then find the next-most promising (p_2, s_2) such that (1) the periods p_1 and p_2 differ by at least 25% of the maximum loop period, and (2) the two loops overlap, i.e., $[s_1, s_1 + p_1) \cap [s_2, s_2 + p_2) \neq \emptyset$.

For each of the two dominant labels (p_i, s_i), we also select the 9 nearest start frames as additional candidates, for 20 labels in total.

The 21st label is for a static frame (period $p = 1$), which is selected as the middle frame of the overlap of the two loops (p_i, s_i).

For the reduced set of 21 labels, the two-stage optimization of [Liao et al. 2013] is unnecessary. We solve a single multilabel graph cut. All pixels are initialized with labels that correspond to the longer of the two loops (p_i, s_i) found above, as we find that the optimization has an easier time changing to the shorter period than vice versa.

As shown later in Table 4, pruning the set of labels reduces looping quality on our example results (objective E' increases from 48.8 to 51.9), but fortunately the change is small.

6.5 Coarse-scale graph cut optimization

As the graph cut is a computational bottleneck, it is important to perform it at the coarse spatial resolution (e.g., 480×270) of $\tilde{V}$. However, at this resolution, we find that the loss of fine-scale detail significantly degrades the estimates of spatiotemporal consistency.

Our solution is to evaluate the consistency objectives at double the spatial resolution of $\tilde{V}$. Figure 11 illustrates the construction for E^*_{spatial}. We define E^*_{temporal} similarly using the sum of fine-scale pixel differences on each block. Effectively, we are solving the problem at a higher resolution (e.g., 960×540) but restricting the labels to have 2×2 spatial granularity.

Agarwala et al. [2005] describe a different multiscale strategy for their 3D multilabel graph cut. They progressively upsample coarser solutions and optimize nodes only in neighborhoods of the seams found at the coarse resolution. Specifically, for each alpha-expansion, they consider only nodes within distance 10 of those that already have the particular alpha label. We initially experimented with a similar strategy for our 2D graph cut and found that the solution is more susceptible to poor local minima inherited from coarse scales. This drawback is discussed in [Agarwala et al. 2005].

6.6 Coarse-scale Poisson blending

The Poisson-blended loop L' must be generated at full resolution. Even with a multigrid scheme, it is daunting to solve a linear system with $3840{\times}2160{\times}150 \approx 1.2$G unknowns (actually, one such system for each of the three color channels).

Similar to Agarwala [2007], we solve for the difference $L_d = L' - L$ rather than L' itself. We briefly review this approach, extending it to the screened Poisson setting in which there is a preference to preserve the original colors in L. Recall that we seek

$$\min_{L'} \|\nabla L' - g\|^2 + \alpha \|L' - L\|^2.$$

A key observation is that the desired gradient g can be expressed as a *sparse* difference g_d from the gradient of the initial loop L,

$$g = \nabla L + g_d,$$

because g_d is nonzero only along the spatiotemporal seams, i.e., at discontinuities of $\phi'(x,t)$.

Therefore the linear system can be represented as

$$(\Delta - \alpha)(L + L_d) = \nabla \cdot (\nabla L + g_d) - \alpha L,$$

which simplifies to

$$(\Delta - \alpha) L_d = \nabla \cdot g_d. \tag{3}$$

Thus, solving for the difference L_d is again a screened Poisson equation, but now with a sparse right-hand-side vector. Due to this sparsity, L_d tends to be low-frequency everywhere except immediately adjacent to the seams. This motivates Agarwala to define L_d using an adaptive quadtree structure.

We use the simple approach of solving (3) on a coarser 3D grid $\bar{L}$. We then upsample the correction $\bar{L}_d$ (using a box filter) and add it to the initial loop L to obtain the blended loop L'. We find that temporal downsampling leads to noticeable artifacts, in part due to the nonuniform time steps in ϕ' resulting from the local temporal scaling of Section 4, so we create the coarse grids $\bar{L}, \bar{g}, \bar{L}_d$ by downsampling just spatially, to the same spatial resolution as $\tilde{V}$. Using a *single* multigrid V-cycle on the coarse-grid system, the fine-scale rms error for Full HD video is only about 0.8%, and the result is visually adequate. The use of a box filter could give rise to 2D blocking artifacts. We had initially implemented Gauss-Seidel relaxation at the resolution of $\tilde{L}$ for this concern, but found that as long as the 3D grid is subsampled only spatially, the 2D blocking artifacts are not noticeable in practice.

7 Reduced memory usage

Storing a 3840×2160 30fps 5-second video using 4 bytes/pixel requires 5.0 GB memory. We use the NV12 representation (half-resolution chroma) to reduce this to 1.9 GB. The input video V is immediately downsampled as it is streamed in, and the output loop L' is also generated and compressed in streaming fashion. Nearly all computation involves downsampled data.

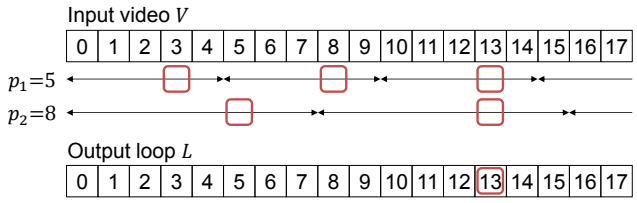


Figure 12: *Because we use two candidate periods, the content at any pixel x of an output frame t is selected (based on (p_x, s_x)) from only a small number of input frames. Here, output frame 13 only needs access to input frames 3, 5, 8, and 13.*

Thus the memory bottleneck is the computation of the blended loop $L' = L + \text{Upsample}(L_d)$, because loop $L(x,t) = V(x, \phi'(x,t))$ needs random-access to the full-resolution frames of the input video. At first, it would appear that the entire input video V must be memory-resident.

However, because we constrain the candidate labels to just two dominant periods p_1, p_2, the content needed to generate $L(\cdot, t)$ comes from only a small number $\{k_1\}, \{k_2\}$ of input frames: $0 \le (t \bmod p_1) + k_1 p_1 < T$ and $0 \le (t \bmod p_2) + k_2 p_2 < T$ (refer to Figure 12). Therefore, an efficient solution is to advance *multiple simultaneous read streams* on the input video V such that only these frames are memory-resident for each output frame t. When processing a 5-second UHD video, the use of multiple read streams reduces maximum memory use from 3.4 GB to 2.1 GB. We were hoping to see a bigger savings; it seems that the use of Windows Media Foundation for video processing introduces a significant memory overhead per read and write stream. On the other hand, this per-stream overhead is independent of the video length.

8 Results and discussion

Figure 13 shows results for 24 example loops, and Table 1 lists some of their properties. The percentage of pixels classified as nonloopable varies greatly depending on the content, as do the dominant periods p_1, p_2 identified in the preprocess and the fraction of pixels assigned to be static. One column analyzes the improvement in dynamism (fraction of looping pixels) provided by the relaxed constraints of the blend-aware consistency metric. As a measure of the uniformity of assigned loop parameter labels, we report the entropy of their distribution over all pixels. The computation times are obtained on a 3.2GHz quad-core Intel Xeon W3565 (acquired in 2009). These times vary only slightly with video resolution because much of the computation occurs on the coarser-scale video $\tilde{V}$. The resolution-dependent work is the downsampling of the input and the final loop assembly including the summation with the upsampled gradient-domain correction.

Table 2 shows the fraction of time spent in each step of loop creation. The costs of decompressing the input video and compressing the output loop are significant. We omit these two steps when reporting computation times. The computational bottleneck is the graph cut optimization, and the Poisson blending is a close second. As shown in Table 3, our timing results are about two orders of magnitude faster than prior techniques, even while processing videos with higher resolution.

Many of the example datasets are from the work of Liao et al. [2013], and we provide comparisons in the accompanying video. The results show that blend-aware consistency enables greater dynamism (i.e., a greater fraction of looping pixels), and that looping quality is not adversely affected by the acceleration techniques of Section 6.

Video	Resolution	% nonloopable pixels	% static w/ $E_{\text{consistency}}$	% static w/ $E^*_{\text{consistency}}$	Increased dynamism	Periods p_1	p_2	Entropy (p_x, s_x)	Time (sec)
balcony	1920×1080	54.5	1.2	0.3	1.0	8	26	2.48	6.6
bridgebirds	1920×1080	66.3	9.0	3.2	5.8	30	8	2.20	5.9
brink	1920×1080	43.4	0.1	0.1	0.0	21	12	3.02	6.8
floraine	1920×1080	13.7	2.5	1.4	1.1	15	30	3.03	8.4
giant	1920×1080	56.1	11.8	3.1	8.7	30	8	2.52	6.2
grandprismatic	1920×1080	73.6	7.5	6.0	1.5	30	12	1.82	4.8
grass	1920×1080	5.1	0.3	0.0	0.3	30	10	2.66	7.9
harlequin	1920×1080	4.4	14.1	5.1	8.9	8	30	3.70	7.5
madisonriver	1920×1080	20.7	0.8	0.0	0.7	8	26	3.01	12.0
morningsteam	1920×1080	61.2	9.7	2.8	6.9	8	30	2.09	8.0
palmtrees	1920×1080	44.5	3.7	0.9	2.8	18	25	2.92	6.0
pigeons	1920×1080	79.0	3.9	3.8	0.0	16	29	1.40	4.5
pinatas	1920×1080	17.5	1.2	0.1	1.1	24	10	3.17	6.8
poolsea	1920×1080	70.5	1.3	0.4	0.9	8	30	2.00	5.6
rampart	1920×1080	54.0	0.7	0.0	0.7	21	8	2.14	7.5
squareflags	1920×1080	36.4	7.6	3.1	4.5	22	8	3.36	5.8
bluepool	3840×2160	31.9	3.2	0.6	2.6	17	28	3.06	13.6
seabeach	3840×2160	44.2	1.5	0.0	1.5	24	10	2.62	13.5
uwfountain	3840×2160	58.6	0.5	0.1	0.3	24	13	2.16	11.5
varkalapool	3840×2160	27.0	4.7	0.8	3.9	25	10	3.58	12.3
pelicans	960×540	83.9	2.0	1.3	0.7	27	8	1.06	3.3
shanghai	960×540	63.0	4.2	7.1	-2.9	24	18	2.03	4.5
snoqualmiefalls	960×540	61.9	2.2	0.1	2.1	8	22	2.18	4.8
streetlight	960×540	61.9	6.7	2.0	4.7	20	15	2.02	4.5

Table 1: *Results for the examples shown in Figure 13: spatial resolution, percentage of pixels classified as nonloopable, percentage of pixels assigned by the optimization to be static using the traditional consistency metric $E_{consistency}$ and using the blend-aware consistency $E^*_{consistency}$, resulting added dynamism, dominant periods (in frames at 7.5fps), average entropy of loop parameter labels, and computation times.*

Processing step	% Time
Load video V	14 %
Downsample to $\tilde{V}$	11 %
Compute cumulative sum tables	2 %
Find best periods p_1, p_2	1 %
Determine initial loop L	33 %
Generate blended loop L'	24 %
Save compressed video L'	15 %

Table 2: *Breakdown of execution time among the various steps in loop creation.*

Method	Output spatial resolution	Time (sec)
[Schödl et al. 2000]	$\leq 360\times240$	n/a (fast)
[Kwatra et al. 2003]	$\leq 360\times240$	300–3600
[Agarwala et al. 2005]†	6000×1200	3600–25200
[Couture et al. 2011]	$2\times6500\times540$	3600
[Tompkin et al. 2011]†	$\leq 960\times540$	> 75
[Bai et al. 2012]†	$\leq 675\times324$	350–1400
[Joshi et al. 2012]†	$\leq 640\times480$	120-600
[Liao et al. 2013]	960×540	480–600
[Sevilla-Lara et al. 2015]	480×360	7200
Ours, Full HD	1920×1080	5–8
Ours, Ultra HD	3840×2160	11–14

Table 3: *Timing results compared with prior techniques for video loop creation. Methods marked '†' rely on some user assistance.*

To quantify the benefits of our individual techniques in terms of improving quality and speed, Table 4 reports the objective function E' and computation times for our final method and with each technique disabled. Recall that E' measures gradient-domain consistency of the output loop and is therefore a better predictor of visual quality than the objective E^* used in the optimization.

Scheme	Objective E'	Time (sec)
Complete method	51.9	8.0
Without Poisson blending	595.0	5.2
Without blend-aware consistency	64.7	6.4
Without promotion of longer loops	55.2	7.6
Without any of the quality improvements	318.0	4.5
Without masked graph cut	71.0	12.5
Without candidate label pruning	48.8	87.8
Without coarse-scale Poisson blend	–	193.2
Without coarse-scale graph cut or any blend	45.4	61.0
Without any of the acceleration techniques	42.6	5785.7

Table 4: *Effects of the techniques from Sections 5 and 6 on quality and speed, measured using median values across the 16 Full HD examples in Table 1.*

We evaluate this gradient-domain consistency at the resolution of the input video V, rather than the resolution of the downsampled video $\tilde{V}$ at which E^* is optimized. One caveat is that the coarse-scale Poisson blending approximation generates uniform small errors in E' which obscure the analysis, so for the purpose of comparing E' values we evaluate Poisson blending accurately at the full resolution (and hence the missing number in the associated row).

The measured consistency values E' corroborate the observed visual improvements, showing that each technique of Section 5 helps overall loop quality.

The bottom half of the timing results in Table 4 reveal the speedup factors of the techniques from Section 6: approximately 1.6 for masked graph cut, 10 for candidate label pruning, 25 for coarse-scale Poisson blend approximation (relative to full-resolution multigrid with ten V-cycles), and 12 for coarse-scale graph cut solution. The speedup factors are not completely independent, so the overall improvement is about 720 instead of 4800.

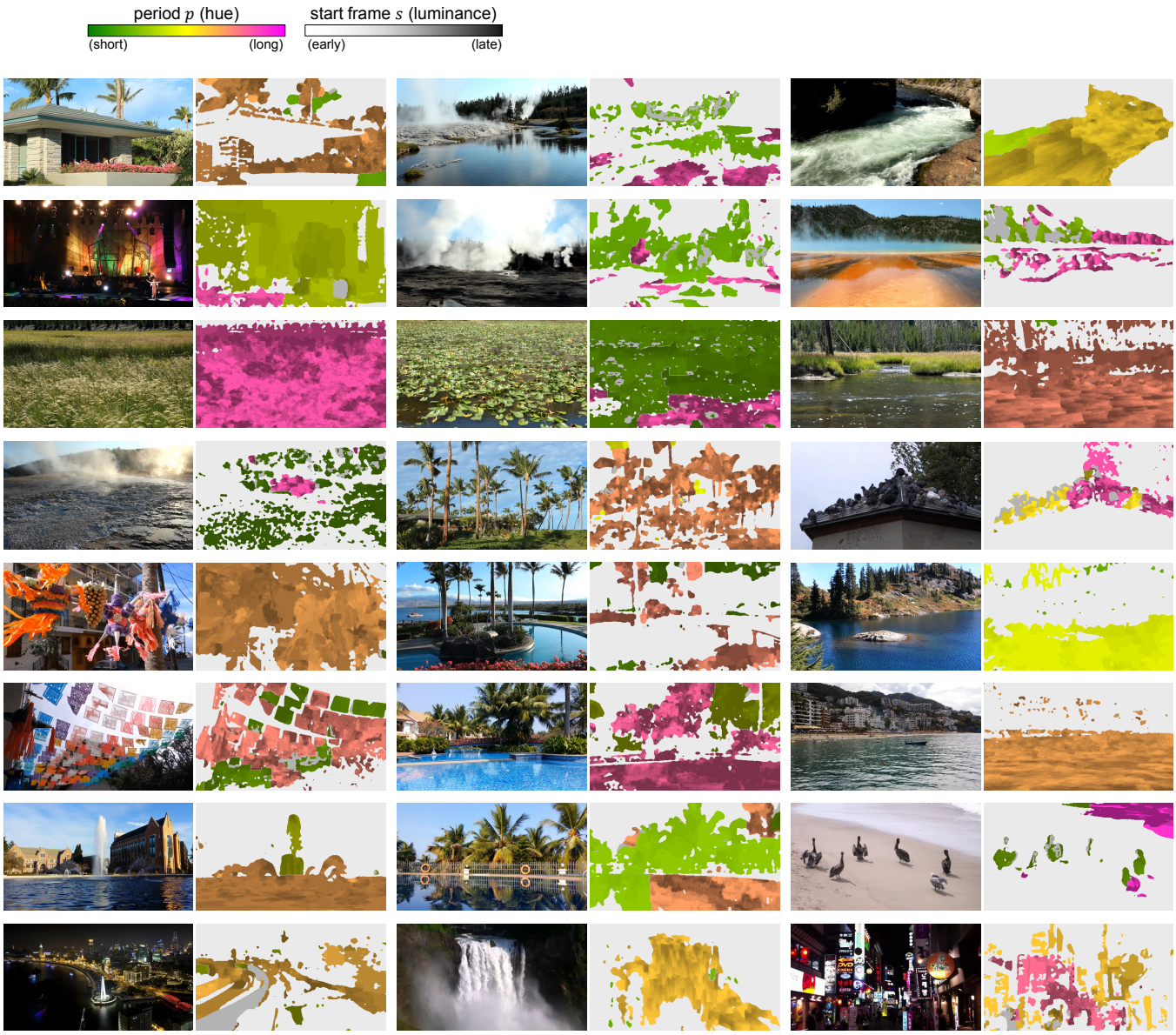


Figure 13: *Example results, showing a frame of the video loop and the associated looping parameters. The loop periods are indicated using hue (green to yellow to red for increasing periods) and start frames using luminance (brighter for later frames). Pixels classified nonloopable are shown white, and pixels assigned static by the optimization are gray.*

A surprising result is that the masked graph cut not only accelerates the computation, it also improves the quality of the resulting loop. The reason is that excluding the nonloopable pixels when estimating the dominant periods provides a better set of candidate labels.

Limitations We inherit many of the limitations demonstrated in prior work on automated video looping. The general approach is most effective on the class of videos exemplified in the results, namely stationary views of natural scenes with organic motions. We observe that some natural motions, such as waves breaking on a beach, have periods that exceed the short 5-second input videos we acquire. For these we would have to capture a longer input and allow a longer maximum loop period. Scenes with moving people or distinct objects are often problematic, as these have sharp delineated boundaries and may lack repeating motion. In such cases, our technique may freeze the objects, leaving just a dynamic background. In some respects, this is opposite the effect sought in cinemagraphs [Beck and Burg 2012], where typically a foreground object is subtly animated in front of a static background. Creating effective cinemagraphs generally requires user guidance or controlled environments.

Within the space of nature environments, our contribution is to expand the category of scenes that can be successfully looped without user input, by recognizing that many low-frequency scene changes (e.g., moving clouds, smoke, steam, shadows) can be smoothed away through Poisson blending.

Local temporal scaling (Section 4) can affect the speed of scene motions. To better bound this temporal distortion, we could select the length of the video loop as a function of the two periods p_1 and p_2 identified during construction. In particular, the loop length could be $\max(p_1, p_2)$ or a small multiple of it.

9 Summary and future work

We have presented techniques to improve quality and efficiency when computing seamless video loops. Together these techniques allow computation of higher quality results about two orders of magnitude faster than prior work.

There are several avenues for future work. We assume that the input video is stabilized. It would be interesting to revisit video stabilization in the specific context of loops. Perhaps feature tracking can be made more robust for periodic scene motions, so that these motions are more easily distinguished from camera shake.

For the blend mask B used to attenuate consistency in blendable regions, it would be interesting to explore a generalization of this construction that considers separate B_{spatial} and B_{temporal} masks.

We have designed our technique to only process short input videos and create correspondingly short loops. For some scenes it may be useful to consider longer input sequences to either identify better short loops or form longer loops. Both the computational cost and memory requirement of our technique should scale linearly with the size of the input. In fact, if one considers a fixed number of candidate (loop parameter) labels, the computational cost may increase sublinearly.

The loop computation may now be fast enough to be practical on mobile devices such as smartphones and cameras. It would be interesting to explore whether it can be further accelerated using specialized hardware.

References

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph.*, 26(3):94.

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D., and SZELISKI, R. 2005. Panoramic video textures. *ACM Trans. Graph.*, 24(3).

BAI, J., AGARWALA, A., AGRAWALA, M., and RAMAMOORTHI, R. 2012. Selectively de-animating video. *ACM Trans. Graph.*, 31(4).

BAI, J., AGARWALA, A., AGRAWALA, M., and RAMAMOORTHI, R. 2013. Automatic cinemagraph portraits. *Computer Graphics Forum*, 32(4):17–25.

BECK, J. and BURG, K. 2012. Cinemagraphs. `http://cinemagraphs.com/`.

BHAT, P., CURLESS, B., COHEN, M., and ZITNICK, L. 2008. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. *European Conference on Computer Vision*, pages 114–128.

COUTURE, V., LANGER, M., and ROY, S. 2011. Panoramic stereo video textures. *ICCV*, pages 1251–1258.

JOSHI, N., MEHTA, S., DRUCKER, S., STOLLNITZ, E., HOPPE, H., UYTTENDAELE, M., and COHEN, M. 2012. Cliplets: Juxtaposing still and dynamic imagery. *Proceedings of UIST*.

KOLMOGOROV, V. and ZABIH, R. 2004. What energy functions can be minimized via graph cuts? *IEEE Trans. on Pattern Anal. Mach. Intell.*, 26(2).

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., and BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286.

LIAO, J., JOSHI, N., and HOPPE, H. 2013. Automated video looping with progressive dynamism. *ACM Trans. Graph.*, 32(4).

LIU, J. and SUN, J. 2010. Parallel graph-cuts by adaptive bottom-up merging. In *Proc. CVPR*.

PÉREZ, P., GANGNET, M., and BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph.*, 22(3).

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., and ESSA, I. 2000. Video textures. In *SIGGRAPH Proceedings*, pages 489–498.

SEVILLA-LARA, L., WULFF, J., SUNKAVALLI, K., and SHECHTMAN, E. 2015. Smooth loops from unconstrained video. *Computer Graphics Forum*, 34(4):99–107.

TOMPKIN, J., PECE, F., SUBR, K., and KAUTZ, J. 2011. Towards moment images: Automatic cinemagraphs. In *Proc. of the 8th European Conference on Visual Media Production (CVMP 2011)*.