# Automated Video Looping with Progressive Dynamism

Zicheng Liao
University of Illinois, Urbana-Champaign

Neel Joshi
Microsoft Research

Hugues Hoppe
Microsoft Research

## Abstract

Given a short video we create a representation that captures a spectrum of looping videos with varying levels of dynamism, ranging from a static image to a highly animated loop. In such a progressively dynamic video, scene liveliness can be adjusted interactively using a slider control. Applications include background images and slideshows, where the desired level of activity may depend on personal taste or mood. The representation also provides a segmentation of the scene into independently looping regions, enabling interactive local adjustment over dynamism. For a landscape scene, this control might correspond to selective animation and deanimation of grass motion, water ripples, and swaying trees. Converting arbitrary video to looping content is a challenging research problem. Unlike prior work, we explore an optimization in which each pixel automatically determines its own looping period. The resulting nested segmentation of static and dynamic scene regions forms an extremely compact representation.

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

**Keywords:** video textures, cinemagraphs, progressive video loops

**Links:** DL PDF WEB VIDEO DATA

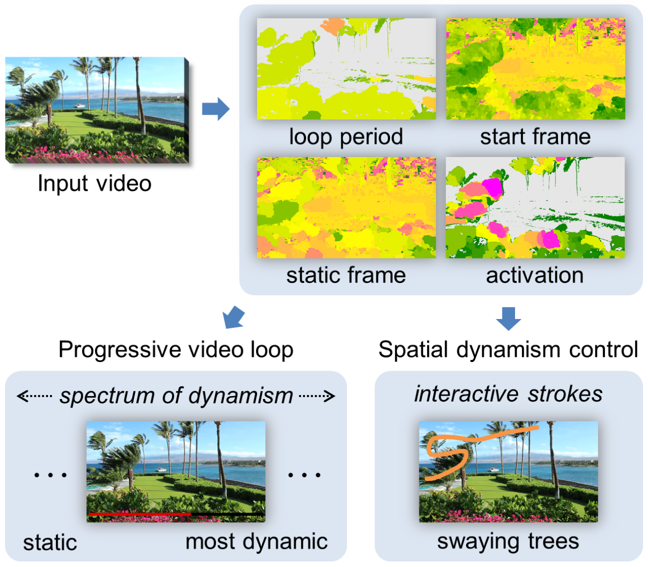

**Figure 1:** *We optimize a set of video looping parameters. This compact encoding defines a spectrum of loops with varying activity and enables fast local control over scene dynamism.*

## 1 Introduction

Many mobile devices now acquire high-definition video just as easily as photographs. With increased parallel processing, the gap in resolution between these two media is narrowing. It should soon become commonplace to archive short bursts of video rather than still frames, with the aim of better capturing the "moment" as envisioned by Cohen and Szeliski [2006].

Several recent techniques explore new ways of rendering short videos. Examples include cinemagraphs [Beck and Burg 2012; Tompkin et al. 2011; Bai et al. 2012] and cliplets [Joshi et al. 2012], which selectively freeze, play, and loop video regions to achieve compelling effects. The contrasting juxtaposition of looping elements against still backgrounds helps grab the viewer's attention. The emphasis in these techniques is on creative control.

We focus on automating the process of forming looping content from short videos. The goal is to render subtle motions in a scene to make it come alive, as motivated by Schödl et al. [2000]. Many such motions are stochastic or semi-periodic, such as swaying grass, swinging branches, rippling puddles, and pulsing lights. The challenge is that these moving elements typically have different looping periods, and moreover some moving objects may not support looping at all. Previous techniques rely on the user to identify spatial regions of the scene that should loop and determine the best period independently for each such region (Section 2).

Instead, we formulate video loop creation as a general optimization in which each pixel determines its own period. An important special case is that the period may be unity, whereby a pixel becomes static. Therefore the optimization automatically segments the scene into regions with naturally occurring periods, as well as regions that are best frozen, to maximize spatiotemporal consistency. A key aspect that makes this optimization more tractable is to parameterize looping content so as to always preserve phase coherence (Section 3).

Our other main contribution is to explore the concept of progressive dynamism (Figure 1). We extend the optimization framework to define a spectrum of loops with varying levels of activity, from completely static to highly animated. This spectrum has a compact encoding, requiring only a fraction of the storage of the input video. We show that this underlying structure also permits local selection of dynamism, for efficient runtime control of scene liveliness based on personal preference or mood. Applications include subtly animated desktop backgrounds and replacements for still images in slide shows or web pages.

Our contributions are:

- Using optimization to automatically segment video into regions with naturally occurring periods as well as static regions.
- Formulating video loop creation using 2D rather than 3D graph cut problems.
- Introducing the progressive video loop, which defines a nested segmentation of video into static and dynamic regions.
- Using the resulting segmented regions to enable interactive, seamless adjustment of local dynamism in a scene.
- Demonstrating an extremely compact encoding.

**ACM Reference Format**
Liao, Z., Joshi, N., Hoppe, H. 2013. Automated Video Looping with Progressive Dynamism. ACM Trans. Graph. 32, 4, Article 77 (July 2013), 10 pages. DOI = 10.1145/2461912.2461950 http://doi.acm.org/10.1145/2461912.2461950.

## 2 Related work

**Video looping** Schödl et al. [2000] define video textures by locating pairs of similar video frames to create a sparse transition graph. A stochastic traversal of this graph generates infinite, non-repeating video. Finding compatible frames may be difficult for scenes with many independently moving elements. Kwatra et al. [2003] synthesize videos using a Markov Random Field (MRF) model. They successively merge video patches offset in space and/or time, and determine the best merging seam using a binary graph cut. Introducing constraints allows the creation of video loops with a specified global period. Agarwala et al. [2005] create panoramic video textures from a panning video sequence. The user selects a static background layer image and draws masks to identify dynamic regions. For each region, the natural periodicity is determined automatically. Then, a 3D MRF problem is solved using a multilabel graph cut on a 3D grid [Boykov et al. 2001]. Couture et al. [2011] create panoramic stereo video textures by blending the overlapping video in the space-time volume.

Although our technique is based on a 3D objective function like these prior graph cut approaches, we define the solution unknowns over a 2D domain grid. Agarwala et al. [2005] discuss that such a 2D problem is as expensive as the 3D version. We show that the cost is reduced by precomputing cumulative-sum tables. The simpler domain lets us introduce looping periods as per-pixel unknowns.

**Cinemagraphs** Tompkin et al. [2011] present a tool for interactive authoring of cinemagraphs. Regions of motion in the video are automatically isolated. The user selects which regions to make looping and which reference frame to use for each region. Looping is achieved by finding matching frames. Bai et al. [2012] describe a method to selectively stabilize motions in video. The user sketches three types of strokes to indicate regions to be made static, immobilized, or fully dynamic. The method propagates the strokes across video frames using optical flow, warps the video to stabilize it, and solves a 3D MRF problem to seamlessly merge it with static content. Applications include cinemagraphs, motion visualization, and video editing. Joshi et al. [2012] develop a set of idioms (static, play, loop, and mirror loop) to let a user quickly combine several spatiotemporal segments from a source video. These segments are stabilized and composited to emphasize scene elements or to form a narrative.

In contrast, our work aims to create video loops without user assistance. Both the spatial regions and their looping periods are found automatically. Looping pixels are allowed staggered start times to improve temporal transitions. A progressive segmentation of the image into static and dynamic regions is formed using a general optimization on spatiotemporal consistency.

**Other video effects** Many other interesting video processing operations have been explored, including animating still images [Freeman et al. 1991; Chuang et al. 2005], magnifying motion [Liu et al. 2005; Wu et al. 2012], enhancing time-lapse video [Bennett and McMillan 2007; Sunkavalli et al. 2007], manipulating time in video editing [Rav-Acha et al. 2007], and creating video synopses [Pritch et al. 2008].

## 3 Basic video loop

The input video is denoted as a 3D volume $V(x, t_i)$, with 2D pixel location $x$ and frame time $t_i$. In forming a video loop, we assume that the input video has already been stabilized. Stabilization can be performed automatically [e.g., Tompkin et al. 2011] or with user guidance [e.g., Bai et al. 2012]. For our results we use either the "Warp Stabilizer" automated tool in Adobe After Effects or a standard feature-based global alignment method as in [Joshi et al. 2012].

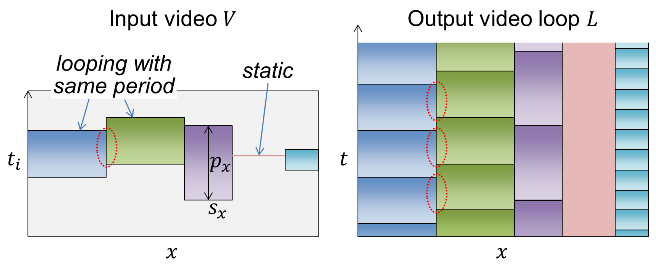

**Figure 2:** *The video loop is specified by assigning each pixel a start time $s_x$ and period $p_x$. Static pixels have $p_x = 1$. The time-mapping function locks phase to help maintain spatial consistency across pixels with the same period, as shown by the red ellipses.*

The goal is to construct an infinitely looping output video $L(x, t)$ with good spatiotemporal consistency, i.e., the loop should avoid undesirable spatial seams or temporal pops which occur when its content is not locally consistent with the input video. Because the input is stabilized, we form $L$ by retrieving for each pixel some content associated with the same pixel in the input, as illustrated in Figure 2. This content may be either *static* or *looping*. In either case, it is represented as a temporal interval $[s_x, s_x + p_x)$ from the source video, where $s_x$ is the start time and $p_x$ is the period, in number of frames. A pixel assigned to be static thus corresponds to the case $p_x = 1$.

More precisely, the output video loop is

$$L(x, t) = V\big(x, \phi(x, t)\big), \quad t \geq 0,$$

where the time-mapping $\phi$ is defined from the unknowns $s_x, p_x$ as

$$\phi(x, t) = s_x - (s_x \bmod p_x) + (t \bmod p_x) + \begin{cases} 0 & \text{if } (t \bmod p_x) \geq (s_x \bmod p_x) \\ p_x & \text{otherwise.} \end{cases} \quad (1)$$

The complicated modulo arithmetic in this formula deserves further explanation. Intuitively, if two adjacent pixels are looping with the same period, it is usually desired that they be *in-phase* in the output loop. Figure 3 illustrate this important point. Two adjacent pixels $x$ and $z$ have the same looping period $p_x = p_z$ and retrieve content from input intervals $[s_x, s_x + p_x)$ and $[s_z, s_z + p_z)$ respectively. Although their start times $s_x, s_z$ differ, their input time intervals have significant overlap. By wrapping these intervals in the output timeline using Equation 1, proper adjacency is maintained within the temporal overlap, and therefore spatial consistency is automatically preserved.

It is interesting to contrast this loop parameterization with that presented by Agarwala et al. [2005, Fig. 5], which solves for time offsets between output and input videos. We instead assume these offsets are prescribed by phase coherence and solve for start frames (Figure 2). As shown later in the results section, good video loops often have regions looping in-phase with a common optimized period but with many staggered per-pixel start times.

Video transitions using graph cut textures [Kwatra et al. 2003] also maintain phase coherence, but they require at least one frame of the output video to be copied in whole from the input video. In contrast, our approach combines unconstrained temporal intervals from an input video, as illustrated in Figure 4.

### 3.1 Construction overview

Prior methods define a 3D MRF problem over a fixed spatiotemporal output domain. In our setting however, modifying per-pixel looping periods has the effect of shifting spatiotemporal adjacencies in the

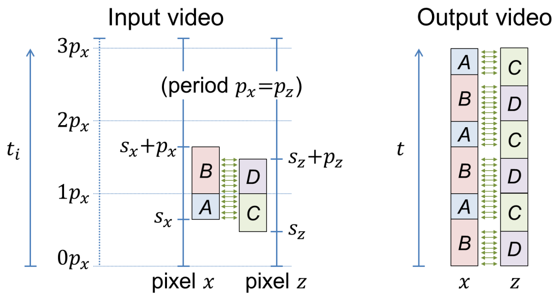

**Figure 3:** *For adjacent pixels $x$ and $z$ with the same looping period $p_x = p_z$ and similar start times $s_x, s_z$, the in-phase time-mapping function of Equation 1 automatically preserves spatiotemporal consistency over a large portion of the output timeline. The green arrows show corresponding spatially adjacent pixels.*

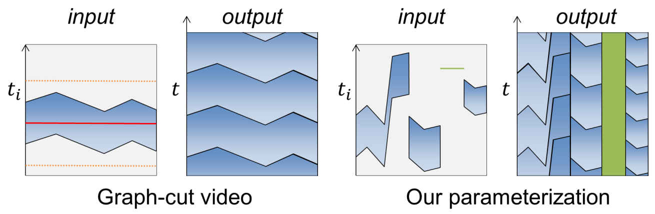

**Figure 4:** *Graph cut video transitions assume the loop has a given global period and contains at least one full input frame (here shown in red), so the search space is bounded (orange dashes). To find compatible content, our approach considers many looping periods and unconstrained time intervals.*

output video. Therefore, we must formulate video loop construction as an MRF problem over the 2D spatial domain rather than the full 3D video volume.

The goal is to find start times $\mathbf{s} = \{s_x\}$ and periods $\mathbf{p} = \{p_x\}$ that minimize the objective

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

The first term encourages all pixel neighborhoods in the video loop to be consistent both spatially and temporally with those in the input video (Section 3.2). The second term penalizes the assignment of static loop pixels except in regions of the input video that are truly static (Section 3.3).

## 3.2 Spatiotemporal consistency

In the generated video loop, each pixel's spatiotemporal neighbors should look similar to those in the input [Agarwala et al. 2005]. Because the domain graph is defined on the 2D spatial grid, the objective must distinguish spatial and temporal consistency:

$$E_{\text{consistency}}(\mathbf{s},\mathbf{p}) = \beta\, E_{\text{spatial}}(\mathbf{s},\mathbf{p}) + E_{\text{temporal}}(\mathbf{s},\mathbf{p}).$$

(We set the parameter $\beta = 10$ for all results.)

**Spatial consistency** The term

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

$$\Psi(x,z) = \frac{1}{T}\sum_{t=0}^{T-1}\left(\begin{matrix}\|V(x,\phi(x,t)) - V(x,\phi(z,t))\|^2 + \\ \|V(z,\phi(x,t)) - V(z,\phi(z,t))\|^2\end{matrix}\right)$$

measures compatibility for each pair of adjacent pixels $x$ and $z$, according to their time mappings $\phi(x,t)$ and $\phi(z,t)$, summed at both pixels $x$ and $z$ over all $T$ time frames in the output video loop. For instance, the pixel color $V(x,\phi(x,t))$ is compared with the pixel color $V(x,\phi(z,t))$ expected by its spatial neighbor at the same output time $t$ and vice versa. Here the period $T$ is the least common multiple (LCM) of all per-pixel periods. Equivalently, the objective can be formulated as $\lim_{T\to\infty} E_{\text{spatial}}$, the average spatial consistency over an infinitely looping output video.

Inspired by [Kwatra et al. 2003], the factor

$$\gamma_s(x,z) = 1\Big/\Big(1 + \lambda_s \operatorname*{MAD}_{t_i} \big\|V(x,t_i) - V(z,t_i)\big\|\Big)$$

reduces the consistency cost between pixels when the temporal median absolute deviation (MAD) of their color differences in the input video is large, because inconsistency is then less perceptible. We use MAD rather than variance because it is less sensitive to outliers. (We set $\lambda_s = 100$ in all results.)

For efficient evaluation we distinguish four cases:

**(1)** When pixels $x$ and $z$ are both made static, the energy reduces to

$$\Psi(x,z) = \|V(x,s_x) - V(x,s_z)\|^2 + \|V(z,s_x) - V(z,s_z)\|^2.$$

**(2)** When only pixel $x$ is static, the energy simplifies to

$$\Psi(x,z) = \frac{1}{T}\sum_{t=0}^{T-1}\left(\begin{matrix}\|V(x,s_x) - V(x,\phi(z,t))\|^2 + \\ \|V(z,s_x) - V(z,\phi(z,t))\|^2\end{matrix}\right).$$

For each of the two summed vector norms and for each color channel $c \in \{1,2,3\}$, the sum is obtained as

$$\frac{1}{T}\sum_{t=0}^{T-1}\big(V_c(x,s_x) - V_c(x,\phi(z,t))\big)^2 =$$

$$V_c^2(x,s_x) - \frac{2V_c(x,s_x)}{p_z}\sum_{t_i=s_z}^{s_z+p_z-1} V_c(x,t_i) + \frac{1}{p_z}\sum_{t_i=s_z}^{s_z+p_z-1} V_c^2(x,t_i).$$

We evaluate the two sums above in constant time by precomputing temporal cumulative-sum tables on $V_c$ and $V_c^2$.

**(3)** When both pixels are looping with the same period $p_x = p_z$, the energy reduces to

$$\Psi(x,z) = \frac{1}{p_x}\sum_{t=0}^{p_x-1}\left(\begin{matrix}\|V(x,\phi(x,t)) - V(x,\phi(z,t))\|^2 + \\ \|V(z,\phi(x,t)) - V(z,\phi(z,t))\|^2\end{matrix}\right).$$

Moreover we detect and ignore the zero-valued terms for which $\phi(x,t) = \phi(z,t)$. As illustrated in Figure 3, for the common case where start times are similar, large time intervals (with contiguous sets of green arrows) can thus be quickly ignored.

**(4)** Finally, when the pixels have different looping periods, we should compute the sum using $T = \mathrm{LCM}(p_x,p_z)$. The apparent worst case when the two periods are relatively prime, i.e., $\mathrm{LCM}(p_x,p_z) = p_x p_z$, is in fact computed efficiently by recognizing that

$$\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(a_i - b_j)^2 = \frac{1}{m}\sum_{i=0}^{m-1} a_i^2 + \frac{1}{n}\sum_{j=0}^{n-1} b_i^2 - \frac{2}{mn}\Big(\sum_{i=0}^{m-1} a_i\Big)\Big(\sum_{j=0}^{n-1} b_i\Big),$$

where $a_i = V_c(x,\phi(x,i))$ and $b_j = V_c(z,\phi(z,j))$. We thus reuse the same precomputed cumulative-sum tables from case (2) to evaluate these terms in constant time.

We use this expected squared difference as an approximation even when the periods $p_x, p_z$ are not relatively prime. This approximation provides an important (6×) speedup, and we verified that it does not appreciably affect the quality of results.

**Temporal consistency** The objective term

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

compares for each pixel the value at the loop start and the value right after the loop end, and for symmetry it also compares the value just before the loop start and at the loop end.

Because looping discontinuities are less perceptible when a pixel varies significantly over time in the input video, we attenuate the consistency cost using the factor

$$\gamma_t(x) = 1/\Big(1 + \lambda_t \operatorname*{MAD}_{t_i} \big\|V(x,t_i) - V(x,t_i+1)\big\|\Big),$$

which estimates the temporal variation at the pixel based on the median absolute deviation of successive pixel differences. (We set $\lambda_t = 400$ in all results.)

For any pixel assigned as static (i.e., $p_x = 1$), $E_{\text{temporal}}$ computes the pixel value difference between successive frames and therefore favors pixels with zero optical flow in the source video. While this behavior is reasonable, in practice we find that it prevents interesting moving objects from being frozen in the static image. Instead, we let the temporal energy be zero for any pixel assigned to be static.

For looping pixels, we considered introducing a factor $1/p_x$ that would account for the fact that a shorter loop reveals a temporal discontinuity more frequently in the output. However, this was found undesirable as it over-penalizes loops with small periods relative to longer loops with equal energy.

## 3.3 Per-pixel dynamism prior

If all pixels are assigned to be static from the same input frame, the loop attains perfect spatiotemporal consistency. To penalize this trivial solution, we seek to encourage pixels that are dynamic in the input video to also be dynamic in the loop. We therefore introduce an additional term $E_{\text{static}}$, which adjusts the energy objective based on whether the neighborhood $N$ of each pixel has significant temporal variance in the input video. If a pixel is assigned a static label, it incurs a cost penalty $c_{\text{static}}$, but this penalty is reduced according to the temporal variance of the pixel's neighborhood. Thus we define

$$E_{\text{static}} = \sum_{x|p_x=1} E_{\text{static}}(x), \quad \text{with}$$

$$E_{\text{static}}(x) = c_{\text{static}} \min\Big(1, \lambda_{\text{static}} \operatorname*{MAD}_{t_i} \big\|N(x,t_i) - N(x,t_i+1)\big\|\Big),$$

where $\lambda_{\text{static}} = 100$ and $N$ is a Gaussian-weighted spatiotemporal neighborhood ($\sigma_x = 0.9$, $\sigma_t = 1.2$).

## 3.4 Optimization algorithm

**Traditional graph cut** Our initial approach was to solve the MRF optimization using a standard multilabel graph cut algorithm, where the set of pixel labels is the outer product of candidate start times $\{s\}$ and periods $\{p\}$. However, this approach may produce relatively poor local minima, as shown in the example of Figure 5. The problem is that a graph cut alpha expansion only considers a single new candidate label. As the algorithm tries to change the solution to a possibly better looping period $p$, it must consider a label that pairs this period $p$ with a single start time $s$. This restriction prevents the optimization from jumping to another valley in the energy landscape where the new period is allowed different start times at different pixels. Jumping out of the local minimum would require considering multiple target labels simultaneously.

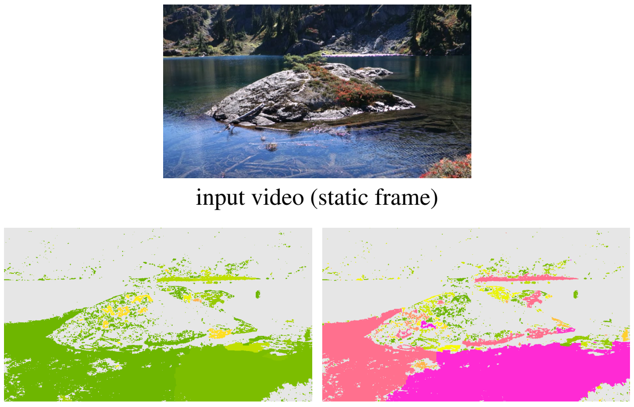

**Figure 5:** *Compared to the traditional graph cut algorithm, our two-stage procedure converges to slightly better minima with longer loop periods. (The green-to-red color map indicates progressively longer periods.) The two-stage algorithm is also about 1.6× faster.*

**Two-stage approach** Instead, we introduce an optimization procedure that works in two stages (left portion of Figure 6):

(1) For each candidate looping period $p > 1$, we find the per-pixel start times $s_{x|p}$ that create the best video loop $L|p$ with just that period, by solving a multilabel graph cut.

(2) Next we solve for per-pixel periods $p_x \geq 1$ that define the best video loop $(p_x, s_{x|p_x})$ using the pixel start times obtained in stage 1, again by solving a multilabel graph cut. Here the set of labels includes all the periods $p > 1$ considered in the first stage, together with all possible frames $s'_x$ for the static case $p = 1$. Essentially, the optimization merges together regions of $|\{p\}| + |\{s\}|$ candidate loops: the optimized loops found in stage 1 for periods $\{p\}$ plus the static loops corresponding to the frames $\{s\}$ of the input video.

The two-stage optimization is about 1.6× faster, because in stage 1 the multiple graph cuts are trivially parallelized and the spatial consistency cost does not have to consider mismatched periods.

The graph cuts in both stages are solved using the iterative alpha-expansion algorithm of Kolmogorov and Zabih [2004]. This algorithm assumes a regularity condition on the energy function, namely that for each pair of adjacent nodes and any three labels $\alpha, \beta, \gamma$, the spatial cost should satisfy $c(\alpha,\alpha) + c(\beta,\gamma) \leq c(\alpha,\beta) + c(\alpha,\gamma)$. However, this constraint is not guaranteed in stage 2. One reason is that when two adjacent pixels are assigned the same period, the fact that they may have different start times means that their spatial cost $c(\alpha,\alpha)$ may be nonzero. Fortunately, the start times are solved in stage 1 to minimize this cost, so it is likely small.

Because the regularity condition does not hold, the theoretical bounded-approximation guarantees of the alpha expansion algorithm no longer hold. Nonetheless, several researchers have reported good results in this case [e.g., Kwatra et al. 2003; Agarwala et al. 2004]. The workaround is to adjust some edge costs when setting up each alpha expansion pass. Specifically, we add small negative costs to the edges $(\beta,\gamma)$ such that the regularity condition is satisfied.

Another reason that the energy function is irregular is that $E_{\text{spatial}}(x,z)$ is a squared distance rather than a Euclidean distance. We find that introducing the square root yields inferior results.

Because the iterative multilabel graph cut algorithm may find only a local minimum of the objective function, the initial state is important. For stage 1, we initialize $s_x$ to minimize only the temporal cost, and for stage 2, we select $p_x$ whose loop $L|p_x$ has lowest spatiotemporal cost at pixel $x$.

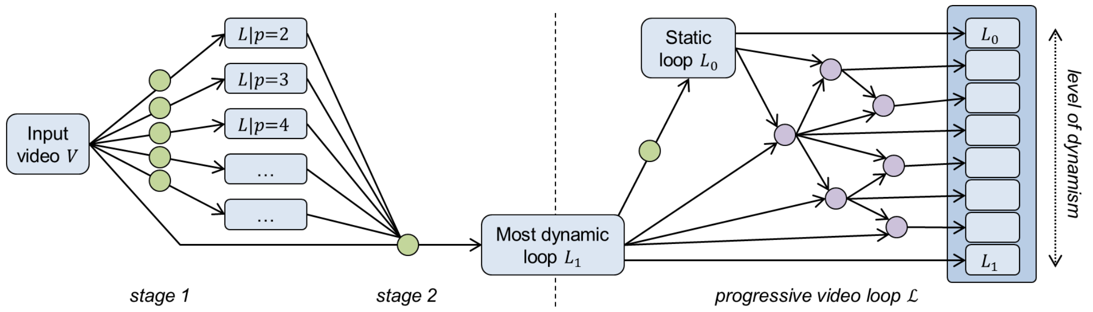

**Figure 6:** *We construct an optimized video loop in two stages: (1) finding optimized loops $L|p$ for each period $p$, and (2) spatially merging these loops together with with static frames of the input video. The green circles denote instances of multilabel graph cuts. Next, creating a progressive video loop involves a recursive partition using fast binary graph cuts, shown as purple circles.*

## 4 Progressive video loops

We now generalize from a single video loop $L$ to a spectrum of loops $\mathcal{L} = \{L_d \mid 0 \le d \le 1\}$ where $d$ refers to *level of dynamism* – a normalized measure of the temporal variance in the video loop (Section 4.2). At one end of the spectrum, the least-dynamic loop $L_0$ corresponds to a static image (where each pixel may be copied from a different frame of the input video). At the other end of the spectrum, the most dynamic loop $L_1$ has the property that nearly all its pixels are looping.[1]

To define the spectrum of loops, we require that each pixel have exactly two possible states:

- *static*, with a color value taken from a single frame $s'_x$ of the input video, or
- *looping*, with a looping interval $[s_x, s_x + p_x)$ that includes the static frame $s'_x$.

We then establish a nesting structure on the set of looping pixels by defining the *activation threshold* $a_x \in [0, 1]$ as the level of dynamism at which pixel $x$ transitions between static and looping.

Thus, a progressively dynamic video loop has the time-mapping

$$\phi_d(x,t) = \begin{cases} s'_x & \text{if } d \le a_x \\ \phi(x,t) & \text{otherwise.} \end{cases}$$

The goal is to determine the activation threshold $a_x$, static frame $s'_x$, loop start frame $s_x$, and period $p_x$ at each pixel such that all video loops in $\mathcal{L}$ have good spatiotemporal consistency.

### 4.1 Construction overview

As shown in Figure 6, our approach consists of three steps:

**(1)** We solve for a **most dynamic** loop $L_1$ using the two-stage algorithm of Section 3 by setting $c_{\text{static}}$ to a large value, $c_{\max} = 10$.

**(2)** We create a **static** loop $L_0$ (i.e., a "reference image") by leaving as-is any pixels already static in $L_1$ and solving for the best static frame $s'_x$ for each remaining pixel (Section 4.3).

**(3)** Having obtained the parameters $(s'_x, s_x, p_x)$ defining the two loops $\{L_0, L_1\} \subset \mathcal{L}$, we assign an **activation threshold** $a_x$ at each pixel to establish the loop spectrum $\mathcal{L}$. This step uses a recursive binary partition over $c_{\text{static}}$ between $L_0$ and $L_1$ (Section 4.4).

[1]Forcing all pixels to loop may introduce bad artifacts for scenes with non-loopable content – artifacts undesirable enough that it is not worth showing these loops to the user.

### 4.2 Parameterization of progressive video loop

The most straightforward way to parameterize the progressive loop spectrum $\mathcal{L}$ is using the static-cost parameter $c_{\text{static}}$ that is varied during construction. However, the level of activity in the loop often changes quite non-uniformly (Figure 8) and differs significantly across videos. We find that a more intuitive parameterization is to use a normalized measure of temporal variance within the loop. Let

$$\operatorname{Var}(L) = \sum_x \operatorname*{Var}_{s_x \le t_i < s_x + p_x} \big(V(x, t_i)\big)$$

measure the temporal variance of all pixels in a video loop $L$. We define the *level of dynamism* as the temporal variance normalized relative to that of the most dynamic loop $L_1$:

$$\operatorname{LOD}(L) = \operatorname{Var}(L) / \operatorname{Var}(L_1).$$

This LOD measure is used to define the dynamism $d$ of each loop $L_d$. Therefore, by definition, the most dynamic loop has $\operatorname{LOD}(L_1) = 1$ and the static loop has $\operatorname{LOD}(L_0) = 0$.

### 4.3 Construction of static loop

To obtain the static loop $L_0$ in step 2, we use the second-stage optimization of Section 3, setting $c_{\text{static}} = 0$, and enforcing the constraint $s_x \le s'_x < s_x + p_x$, as shown in Figure 7. For this step, we also introduce an extra data term that penalizes squared color differences from each static pixel to its corresponding median value in the input video. Encouraging median values helps create a static image that represents a "still" moment, free of transient objects or motions [Wang et al. 2005].

### 4.4 Optimization of per-pixel activation thresholds

The assignment of activation thresholds in step 3 operates as follows. Recall that we obtained the most dynamic loop $L_1$ by setting the parameter $c_{\text{static}}$ to a large value $c_{\max} = 10$ and we obtained the static loop $L_0$ by setting this same parameter to zero. For each pixel $x$

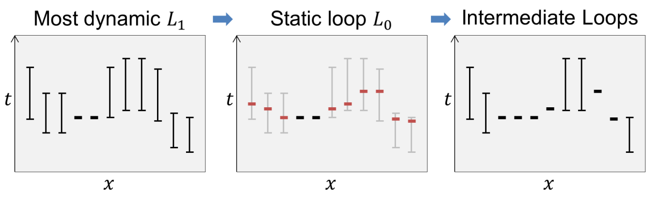

**Figure 7:** *The looping parameters for the most static loop and the subsequent intermediate loops are constrained to be compatible with previously computed loops.*

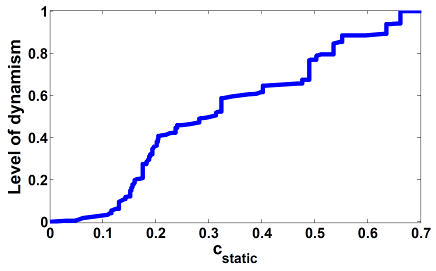

**Figure 8:** *A video loop's level of dynamism is often non-uniform as a function of the optimization parameter $c_{\text{static}}$, and it has step transitions corresponding to the activation of coherent pixel regions.*

looping in the most dynamic loop $L_1$, we know that its transition from static to looping must occur between loops $L_0$ and $L_1$, and therefore its activation threshold satisfies $0 \leq a_x < 1$.

We form an intermediate loop by setting $c_{\text{static}} = (0 + c_{\text{max}})/2$ (halfway between the settings for $L_0$ and $L_1$) and constraining each pixel $x$ to be either static as in $L_0$ or looping as in $L_1$. Minimizing $E$ is thus a *binary* graph cut problem. Let $d$ be the resulting loop's level of dynamism, so the loop is denoted $L_d$. The assignment of each pixel as static or looping in loop $L_d$ introduces a further inequality constraint on its activation threshold $a_x$, i.e., either $a_x < d$ or $a_x \geq d$. We further partition the intervals $[L_0, L_d]$ and $[L_d, L_1]$ recursively to precisely define $a_x$ at all pixels, as illustrated by the cascade of binary graph cuts in Figure 6.

In the limit of recursive subdivision, the activation threshold of each pixel converges to a unique value. Recursion terminates when the change in the static-cost parameter $c_{\text{static}}$ becomes sufficiently small ($< 10^{-6}$) or when the difference between the levels of dynamism of the two loops is sufficiently small ($< 0.01$). As a postprocess, each activation level is adjusted to lie at the midpoint of its vertical step in Figure 8 (rather than at the maximum of the step).

In the spectrum $\mathcal{L}$, there are intervals of dynamism over which the loop does not change, i.e., the vertical steps in Figure 8. Such discontinuities must exist, because the dynamism level is continuous whereas the set of possible loops is finite. The reason that some intervals are large is that maintaining spatiotemporal consistency requires some spatial regions to transition coherently. For some videos this leads to significant jumps in dynamism. To reduce these jumps, we lower the spatial cost parameter $\beta$ from 10 to 5 during step 3; however, the tradeoff is that some of the new transitions are more noticeable.

**Ordering of progressive dynamism** Another issue is that in a progressive video loop, we would prefer that the activation threshold for subtle loops (with less activity) be smaller than that for highly dynamic loops. Unfortunately, with $E_{\text{static}}$ as defined in Section 3.3, varying $c_{\text{static}}$ has the opposite effect: when $c_{\text{static}}$ is low (near the static video $L_0$), pixels with high temporal variance benefit from the greatest drop in $E_{\text{static}}$ when they transition to looping, and therefore the most active loops are introduced first. (Conversely, when $c_{\text{static}}$ is high near the most dynamic video $L_1$, only pixels with low temporal variance have sufficiently small $E_{\text{static}}$ penalty to become static, and therefore the least active loops are introduced late.)

To address this, only for the duration of step 3 we redefine

$$E_{\text{static}}(x) = c_{\text{static}}\Big(1.05 - \min\big(1, \lambda_{\text{static}} \operatorname*{MAD}_{t_i} \big\|N(x,t_i) - N(x,t_i+1)\big\|\big)\Big).$$

Because the loops $L_0, L_1$ bounding the recursive partition process are fixed, the only effect is to modify the activation thresholds and thereby improve the ordering in which the loops appear. Please refer to the supplemental video for some example comparisons.

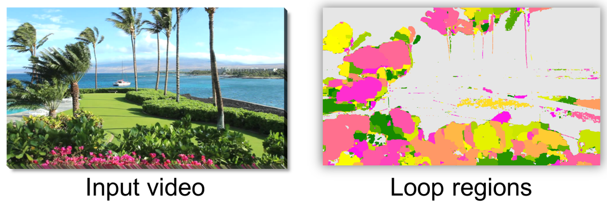

**Figure 9:** *Segmentation of scene into independent looping regions, visualized with random colors and static pixels in light gray.*

### 4.5 Implementation details

In most of our results, the input video is a 5-second segment recorded at 30 frames/sec. To reduce computational cost, we quantize loop start times and periods to be multiples of 4 frames. Also, we set a minimum period length of 32 frames.

As another speedup, to compute the spatial cost between two pixels looping with the same period (case (3) in Section 3.2), we aggregate groups of 4 consecutive pixels into 12-dimensional vectors, and perform principal component analysis (PCA) as a preprocess to project these vectors into an 8 dimensional space. This PCA projection retains over 99% of the data's variance in our tests.

We parallelize the graph cut optimizations using OpenMP and only use a few iterations through all candidate alpha-expansion labels. We find that two iterations are sufficient for the second stage of our optimization and a single iteration is sufficient for all other stages.

## 5 Interactive spatial control over dynamism

The per-pixel periods and activation levels induce a segmentation of the scene into independent looping regions. Whereas the progressive video loop defines a single path through this space, we can also let the user adapt dynamism spatially by selectively overriding the looping state per region.

For fine-grain control, the selectable regions should be small. Yet, they must be sufficiently large to avoid spatial seams when adjacent regions have different states. We place two adjacent pixels in the same region if (1) they share the same looping period, (2) their temporal extents overlap, and (3) they have the same activation level. A flood-fill algorithm finds the equivalence classes for the transitive closure of this relation, as shown in Figure 9.

Our prototype system offers a simple interface to manipulate dynamism over the regions. As the cursor hovers over the video, the local underlying region is highlighted in yellow, and the other regions are shaded with a color code to delineate each region and its current state (shades of red for static and shades of green for looping). Clicking the mouse on the current highlighted region toggles its looping state. Alternatively, dragging the cursor starts the drawing of a stroke. All regions that overlap the stroke are activated or deactivated depending on whether the shift key is pressed. The action is instantaneous, so prior strokes are not retained. Please refer to the supplemental video for examples.

## 6 Local alignment

In some cases, scene motion or parallax can make it difficult to create high-quality looping videos. For these cases, shown in our supplemental video, we optionally perform local alignment of the input video content to enable better loop creation. This is related to the scenario and approach explored by Bai et al. [2012], with a few significant differences. Because their focus is creative control, their approach uses several types of user-drawn strokes to indicate regions to stabilize, keep static, and loop. In contrast, our approach is automatic and does not require any user input, and the creative control (see Section 5) is independent of the alignment process.

Our approach is to treat strong, low-frequency edges as "structural" edges that must be aligned directly and high spatiotemporal frequency areas as "textural" regions whose flow is smoothly interpolated. The visual result is that aligned structural edges appear static, leaving the textural regions dynamic and able to be looped. We achieve this by using a pyramidal optical flow algorithm [Horn and Schunk 1981], with a high degree of smoothing, to align each frame of the video to a reference video frame ($t_{ref}$). We use a 3-level pyramid and the same smoothing constant for all results. The reference frame is chosen as the frame which is most similar to all other frames before local alignment.

Within the optimization algorithm (Section 3), we introduce two additional data terms. The first term

$$E_{\text{aligned}}(x) = \begin{cases} \infty & \text{if } p_x = 1 \text{ and } s_x \neq t_{ref} \\ 0 & \text{otherwise,} \end{cases}$$

forces all static pixels (i.e., not looping) to be taken from the reference frame. The other term

$$E_{\text{flow}}(x) = \begin{cases} 0 & \text{if } p_x = 1 \\ \lambda_f \max\limits_{s_x \leq t_i < s_x + p_x} F(x, t_i) & \text{otherwise,} \end{cases}$$

penalizes looping pixels in areas of low confidence for optical flow, where $F(x, t_i)$ is the flow reprojection error (computed at the next-to-finest pyramid level) for a pixel $x$ at time $t_i$ aligned to the reference frame $t_{ref}$. We set $F(x, t_i) = \infty$ for any pixels where the reprojection error is larger than the error before warping with the flow field or where the warped image is undefined (due to out-of-bounds flow vectors). (We set $\lambda_f = 0.3$ in all results.)

The result of these two terms is that the optimization avoids loops aligned with poor flow and forces regions that cannot be aligned at all to take on values from the static reference frame, which has no alignment error by construction. The looping areas then come from pixels where the flow error at a coarse level of the pyramid is low.

## 7 Rendering

**Crossfading** We apply crossfading to help mask spatial and temporal discontinuities in a video loop, but only where spatial and temporal inconsistencies are significant, so as to avoid excessive blurring elsewhere. We perform temporal crossfading during loop creation, using a linear blend with an adaptive window size that increases linearly with the loop's temporal cost (up to $\pm 5$ frames). Spatial crossfading is performed at runtime using a spatial Gaussian filter $G$ at a subset of pixels $S$. The set $S$ consists of the spatiotemporal pixels with a large spatial cost ($\geq 0.003$), as well as the pixels within an adaptive window size that increases with the spatial cost (up to a $5 \times 5$ neighborhood). For each pixel $x \in S$ we compute

$$L(x, t) = \sum_{x'} G(x' - x)\, V\big(x, \phi(x', t)\big).$$

**Smooth progressive transitions** As the level of dynamism is varied, some fraction of pixels transition from static to looping or vice versa. In either case, it is desirable to maintain spatiotemporal consistency to avoid visible artifacts.

Our initial strategy sought to exploit the fact that the static frame at each pixel lies within the temporal range of its loop. A natural idea is to wait until the (active or potential) loop reaches this frame before transitioning (from or to) the loop. Although this eliminates temporal pops, it has two significant drawbacks. First, it introduce a significant lag in responsiveness. Second, the transition frame often differs at adjacent pixels, resulting in temporary but noticeable spatial inconsistencies. Instead, we simply crossfade between the static and looping states over an interval of 20 frames.

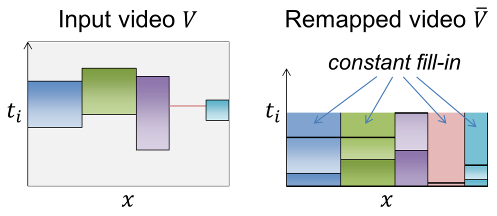

**Figure 10:** *The accessed portion of the input video is remapped to a shorter video $\bar{V}$, whereby the last value of each pixel's loop is held constant.*

## 8 Compression

Besides the input video $V$, the progressive loop representation $\mathcal{L}$ needs only four per-pixel parameters $(s'_x, s_x, p_x, a_x)$. These parameters have strong spatial coherence as visualized in Figure 11. We store the parameters into a four-channel PNG images, with activation thresholds $a_x$ quantized to 8 bits. As shown in Table 1, the compressed representation is about 200 KB per video, a small fraction of the video itself.

Because the progressive loop spectrum $\mathcal{L}$ accesses only a subset of the input video, we repack this content into a shorter video $\bar{V}$ of $\max_x p_x$ frames, by evaluating the initial frames of the most-dynamic loop $L_1$:

$$\bar{V}(x, t_i) = V\big(x, \phi_1(x, t_i)\big), \quad 0 \leq t_i < \max_x p_x.$$

The time-mapping function is then extremely simple:

$$\bar{\phi}(x, t) = t \bmod p_x.$$

Note that it becomes unnecessary to store the loop start times $s_x$, which have high entropy. The static frames are adjusted as $\bar{s}'_x = \bar{\phi}(x, s'_x)$. We reduce entropy in unused portions of the shortened video by freezing the last pixel of each loop (Figure 10). As shown by the compression results in the right columns of Table 1, this remapped video representation ($\bar{V}$ together with $\bar{s}'_x, p_x, a_x$) achieves significant space savings.

## 9 Results and discussion

Our procedure for constructing progressive video loops is fully automatic, and we have tested it on over a hundred videos. This includes a variety of casually shot videos acquired by us (filmed using handheld or tripod-mounted cameras and smartphones) and videos from five related works [Schödl et al. 2000; Kwatra et al. 2003; Tompkin et al. 2011; Bai et al. 2012; Joshi et al. 2012]. We enable local alignment for a few of the videos from Bai et al. [2012].

For our tests, we limit input videos to 5 seconds or less at up to 30 frames/second. We compute most results at a resolution of 960×540 pixels, downsampling the input video if necessary. As shown in Table 1, the processing time for videos of this size is about 8–10 minutes. All our examples are computed on a 2.5GHz Intel Xeon L5420 (2 processor) PC with 16 GB of memory.

In supplemental material we include video results at HD resolution (1920×1080). These are obtained by upsampling the looping parameters computed at the lower resolution (with nearest-neighbor interpolation) and using the resulting time-mapping on the HD input.

| Name | Figure | Resolution | Time (min) | Compressed (KB) Video $V$ | Compressed (KB) $(s'_x, s_x, p_x, a_x)$ | Compressed remapped (KB) Remapped $\bar{V}$ | Compressed remapped (KB) $(\bar{s}'_x, p_x, a_x)$ | Compression ratio |
|---|---|---|---|---|---|---|---|---|
| Pool | 11a | 960x540 | 10 | 4,415 | 189 | 1,930 | 69 | 0.43 |
| Drummers | 11b | 960x540 | 9 | 1,952 | 202 | 532 | 90 | 0.29 |
| Waterwheel | 11c | 960x540 | 9 | 4,822 | 183 | 3,182 | 72 | 0.65 |
| Aquarium | 11d | 960x540 | 8 | 2,065 | 238 | 887 | 98 | 0.43 |
| Flowers | 11e | 960x540 | 9 | 4,386 | 198 | 1,498 | 95 | 0.35 |
| Pinatas | 11f | 960x540 | 10 | 5,165 | 153 | 2,473 | 66 | 0.48 |
| Flags | 11g | 960x540 | 10 | 3,780 | 164 | 1,585 | 59 | 0.42 |
| Citylights | 11h | 960x540 | 6 | 1,767 | 192 | 736 | 75 | 0.41 |

**Table 1:** *Timing and compression results. The optimized parameters (static frame $s'_x$, loop start frame $s_x$, period $p_x$, and activation threshold $a_x$) are encoded into a PNG file. Remapping the video significantly reduces size while preserving the ability to adapt dynamism.*

Figure 9 shows a selection of eight results using a diverse set of input videos. We show the static image corresponding to the least-dynamic loop $L_0$ and the per-pixel looping parameters: the period $p_x$, start frame $s_x$, and activation parameter $a_x$. Note how these parameter maps capture and encode a spatiotemporal segmentation of the video. Please see our supplemental video to appreciate these results and to see more examples.

The results show that there are generally a few dominant periods, which are a function of the scene's motion; however, within one period there are often many staggered per-pixel start times. We have found that having both per-pixel periods and start times greatly increases the quality of the results, as demonstrated in our supplemental video.

Our supplemental video also shows how a user can explore a spectrum of dynamic imagery by interactively varying the level of dynamism of the looping video. The extremes of the spectrum produce a static image or a fully looping video, respectively, while values in the middle produce cinemagraph-type imagery. We also show how a user can override the prioritization encoded in the activation parameter, by changing the looping state of regions using mouse clicks and strokes.

Lastly, our supplemental video shows results with data from five previous works [Schödl et al. 2000; Kwatra et al. 2003; Tompkin et al. 2011; Bai et al. 2012; Joshi et al. 2012] along with side-by-side comparisons. The first two are automatic schemes like ours, and on their datasets our results are similar in quality or have slightly fewer artifacts (especially blurring and ghosting). The next three schemes require user interaction, and our results are comparable in quality while requiring no interaction. Of course, without a user in the loop, we do not achieve the same semantics. Some of the videos of Bai et al. are challenging due to large motions; where our automated alignment fails we still produce results with good quality but different semantics. Possibly our results could be improved with a more sophisticated optical flow method.

Compared to the approaches of Schödl et al. [2000] and Kwatra et al. [2003], our framework sacrifices flexibility for performance by encoding only a single, contiguous loop per pixel. Nonetheless, our results are quite good. Notably, having fewer temporal transitions leads to larger spatiotemporal regions that are coherent with the input video and are therefore automatically free of inconsistencies. Our contribution is to extend beyond these textures to more general videos consisting of many different moving elements as well as static and non-loopable elements, and to free the user from having to select a specific input interval and looping period. Achieving high-quality transitions comes largely from the fact that we optimize both loop periods and start times at each pixel. What we lose in temporal variety by having only a single loop per-pixel, we gain by having different loop periods across pixels. In other words, having multiple loops as in video textures [Schödl et al. 2000] may be more important for a perception of variety when the loop is global across the image.

## 10 Conclusions and future work

In this work, we presented a method for creating a spectrum of looping videos from short input videos. As a result of per-pixel optimization of the loop period and start time, our method can create a high-quality, fully looping video without any user input. An added benefit is the ability to find many subtle but interesting loops that are not easy to discover manually. Our optimization also models scene liveliness and creates a segmentation that encodes spatial regions that loop contiguously. Along with this, we compute a default prioritization for transitioning these regions from static to dynamic with minimal visual artifacts.

A user may interactively override this prioritization while still exploiting a segmentation structure that conveniently identifies good independently looping regions. With simple interactions, a user can create looping videos with a desired amount of scene liveliness. Our optimization is fairly efficient and produces results that are highly compressible. We have addressed many challenges in creating progressively dynamic videos; however, our results also suggest several areas for future work.

**Limitations** In some cases our optimization produces results that have no perceivable artifacts, yet the semantic relationship between objects is disturbed. Post-hoc user interaction can compensate for many of these semantic errors. An interesting opportunity for future work is to provide lightweight interaction for steering the optimization directly.

Stabilization errors can introduce subtle unintended motion which is interpreted as scene motion and is therefore captured in a video loop. Stabilization is an active research area, so future improvements can help reduce these artifacts.

Some slowly moving scene elements, such as smoke or steam, cannot form good loops within the longest periods we consider. Looping these elements would require a longer input video and a more efficient optimization strategy such as a hierarchical solver [Agarwala et al. 2005].

Our current discrete subsampling of periods (multiples of four) can create suboptimal loops. This might be overcome by fine tuning loop periods and start times using a fast postprocess.

Residual seams are sometimes visible for elements with low-frequency spatial content such as clouds. In these cases, the spatial support of our current crossfading is too narrow to adequately diffuse the discontinuities. A Laplacian or Poisson blend [Burt and Adelson 1983; Mahajan et al. 2009] could be used instead.

Although our method adapts to diverse video content and thus minimizes the need for "parameter tweaking", there are cases where adjusting the spatial cost parameter $\beta$ improves results. The fundamental reason is that "texture-like" and "object-like" content have different gradient distributions. We believe it should be possible to learn a data-dependent function for $\beta$ by leveraging ideas from content-adaptive image priors [Cho et al. 2010].

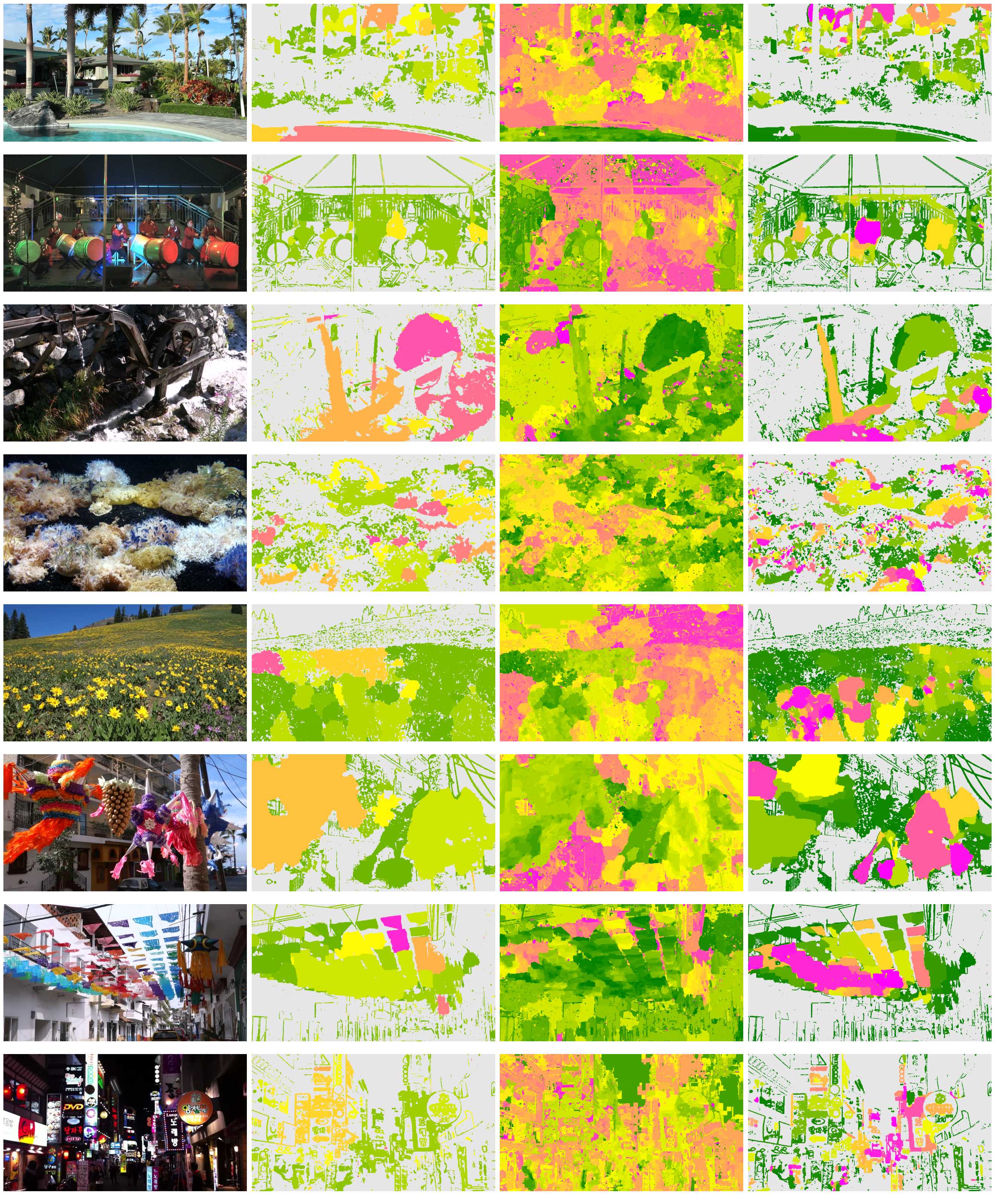

**Figure 11:** *Progressive video loop results, showing the static image corresponding to the least-dynamic loop $L_0$ and the per-pixel looping parameters. All parameters are visualized using a colormap ranging from green through yellow to red as values increase, with light gray indicating static pixels. The activation parameter $a_x$ in the rightmost column encodes the level of dynamism at which each pixel transitions from static to looping. Please refer to the supplemental material to see the progression of video loops.*

**New functionalities** Progressive dynamism could be generalized to control motion amplitude [Wu et al. 2012] rather than just atomically turning it on and off. Another improvement is to anticipate subsequent spatiotemporal feathering when estimating temporal cost near mismatched features. We know that spatiotemporal feathering reduces visible artifacts, but currently our optimization does not solve for labels that will give the optimal feathered results. Lastly, we could incorporate optical flow in the spatiotemporal consistency term to ensure more consistent motion between neighboring pixels.

For new user-facing features, we are interested in automatic creation of mirror loops [Joshi et al. 2012] in addition to ordinary loops. We would also like to use a design gallery [Marks et al. 1997] type of approach to let a user quickly select from an interesting set of automatically created results with different levels of dynamism or different static and looping regions.

## Acknowledgments

We thank Sing Bing Kang for helpful discussions, David Forsyth for his support, the reviewers for their attentive comments, and the authors of previous work for making their data freely available.

## References

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

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).

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

BENNETT, E. P. and MCMILLAN, L. 2007. Computational time-lapse video. *ACM Trans. Graph.*, 26(3).

BOYKOV, Y., VEKSLER, O., and ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 23(11).

BURT, P. J. and ADELSON, E. H. 1983. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2(4).

CHO, T. S., JOSHI, N., ZITNICK, C. L., KANG, S. B., SZELISKI, R., and FREEMAN, W. T. 2010. A content-aware image prior. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

CHUANG, Y.-Y., GOLDMAN, D. B., ZHENG, K. C., CURLESS, B., SALESIN, D. H., and SZELISKI, R. 2005. Animating pictures with stochastic motion textures. *ACM Trans. Graph.*, 24(3).

COHEN, M. and SZELISKI, R. 2006. The moment camera. *IEEE Computer*, 39(8).

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

FREEMAN, W. T., ADELSON, E. H., and HEEGER, D. J. 1991. Motion without movement. *ACM SIGGRAPH Proceedings*.

HORN, B. K. P. and SCHUNK, B. G. 1981. Determining optical flow. *Artificial Intelligence*, 17:185–203.

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.

LIU, C., TORRALBA, A., FREEMAN, W. T., DURAND, F., and ADELSON, E. H. 2005. Motion magnification. *ACM Trans. Graph.*, 24(3):519–526.

MAHAJAN, D., HUANG, F.-C., MATUSIK, W., RAMAMOORTHI, R., and BELHUMEUR, P. 2009. Moving gradients: A path-based method for plausible image interpolation. *ACM Trans. Graph.*, 28(3):42.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., and SHIEBER, S. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. *ACM SIGGRAPH Proceedings*.

PRITCH, Y., RAV-ACHA, A., and PELEG, S. 2008. Nonchronological video synopsis and indexing. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 30(11).

RAV-ACHA, A., PRITCH, Y., LISCHINSKI, D., and PELEG, S. 2007. Dynamosaicing: Mosaicing of dynamic scenes. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 29(10).

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

SUNKAVALLI, K., MATUSIK, W., PFISTER, H., and RUSINKIEWICZ, S. 2007. Factored time-lapse video. *ACM Trans. Graph.*, 26(3).

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)*.

WANG, J., BHAT, P., COLBURN, R. A., AGRAWALA, M., and COHEN, M. F. 2005. Interactive video cutout. *ACM Trans. Graph.*, 24(3).

WU, H.-Y., RUBINSTEIN, M., SHIH, E., GUTTAG, J., DURAND, F., and FREEMAN, W. 2012. Eulerian video magnification for revealing subtle changes in the world. *ACM Trans. Graph.*, 31(4).