Scene Summarization via Motion Normalization

Scott Wehrwein, Kavita Bala, *Member, IEEE*, Noah Snavely *Member, IEEE*

Abstract—When observing the visual world, temporal phenomena are ubiquitous: people walk, cars drive, rivers flow, clouds drift, and shadows elongate. Some of these, like water splashing and cloud motion, occur over time intervals that are either too short or too long for humans to easily observe. High-speed and timelapse videos provide a popular and compelling way to visualize these phenomena, but many real-world scenes exhibit motions occurring at a variety of rates. Once a framerate is chosen, phenomena at other rates are at best invisible, and at worst create distracting artifacts. In this paper, we propose to automatically *normalize* the pixel-space speed of different motions in an input video to produce a seamless output with spatiotemporally varying framerate. To achieve this, we propose to analyze scenes at different *timescales* to isolate and analyze motions that occur at vastly different rates. Our method optionally allows a user to specify additional constraints according to artistic preferences. The motion normalized output provides a novel way to compactly visualize the changes occurring in a scene over a broad range of timescales.

1 Introduction

The increasing ubiquity of video cameras, sharing platforms, and processing power have driven renewed interest in exciting applications of video, from timelapse apps to hyperlapse [1], [2] and virtual reality capture. The popularity of these applications is apparent from a simple search of a video sharing site like Vimeo or YouTube. In particular, anyone from an amateur with a smartphone to a professional artist with expensive motion stages can set up a timelapse capture to create a compelling visualization of changes in the world that occur too slowly to watch in real-time.

One challenge when capturing a timelapse video is deciding the framerate, i.e., how much to speed up time. For example, in the scene in Figure 1, a speedup of 256x (relative to 30 frames per second) is a good choice to illustrate the motion of the clouds; meanwhile, the tree's shadow and the minute hand on the clock move even slower, and are better viewed at 1024x. However, the people walking along the path are aliased at either of these rates, producing strobing artifacts and distracting a viewer from the interesting longer-term phenomena. To convey the details of their motion, the people should be displayed at a real time rate.

It is impossible to achieve all of these goals at once with a standard timelapse video, which is limited to a single framerate. Even if the framerate is allowed to vary over time, we cannot show all of these effects because they occur simultaneously. In this work, we visualize dynamic phenomena at multiple timescales in a single video with spatiotemporally varying framerate. Figure 2 illustrates our method; we begin by decomposing a long video of a scene into a *temporal pyramid* to allow for analysis of phenomena at multiple timescales. We then estimate motion in each timescale using optical flow and feature tracking, and finally use graph cuts and Poisson blending to synthesize a seamless *motion-normalized* output video that simultaneously shows changes unfolding at multiple timescales. Because there may be many ways to choose content from different timescales, we also allow a user to guide the process according to artistic preference by specifying additional constraints.

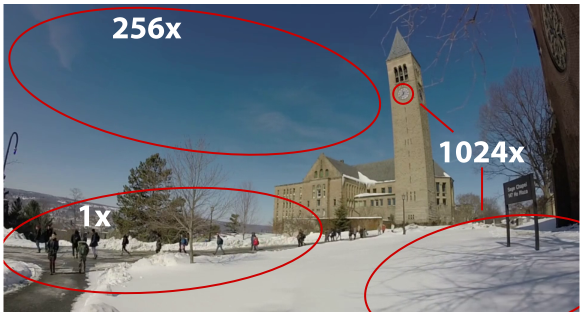


Fig. 1: Different portions of this scene are best viewed at different framerates. People move in real time, whereas clouds move more slowly; the tree's shadow and the clock need an even greater speedup for their motion to be apparent.

To summarize, we make three primary contributions:

1) We demonstrate an automatic *motion normalization* method that combines dynamic scene content from multiple timescales into a single seamless video.
2) We propose to use *temporal pyramids* as a vehicle for the analysis of motions occurring at multiple timescales in a scene.
3) We present a user-in-the-loop variant of our method that allows for expression of artistic preferences.

A successfully motion-normalized output video can provide a more complete illustration of the motions occurring in a scene over time. We demonstrate motion normalization results for regular and high speed video of scenes containing a wide range of phenomena, including waves, clouds, pedestrians, clocks, waterfalls, sun, shadows, kites, flags, geysers, mist, and sunbeams, among others.

2 Related Work

A large body of existing work deals with analyzing, manipulating, and summarizing motion and time in videos.

- *Scott Wehrwein is with Western Washington University, Bellingham, WA. E-mail: scott.wehrwein@wwu.edu*
- *Kavita Bala is with Cornell University, Ithaca, NY.*
- *Noah Snavely is with Cornell Tech, New York, NY*

Manuscript received July DD, 2019; revised MMMMM DD, 20YY.

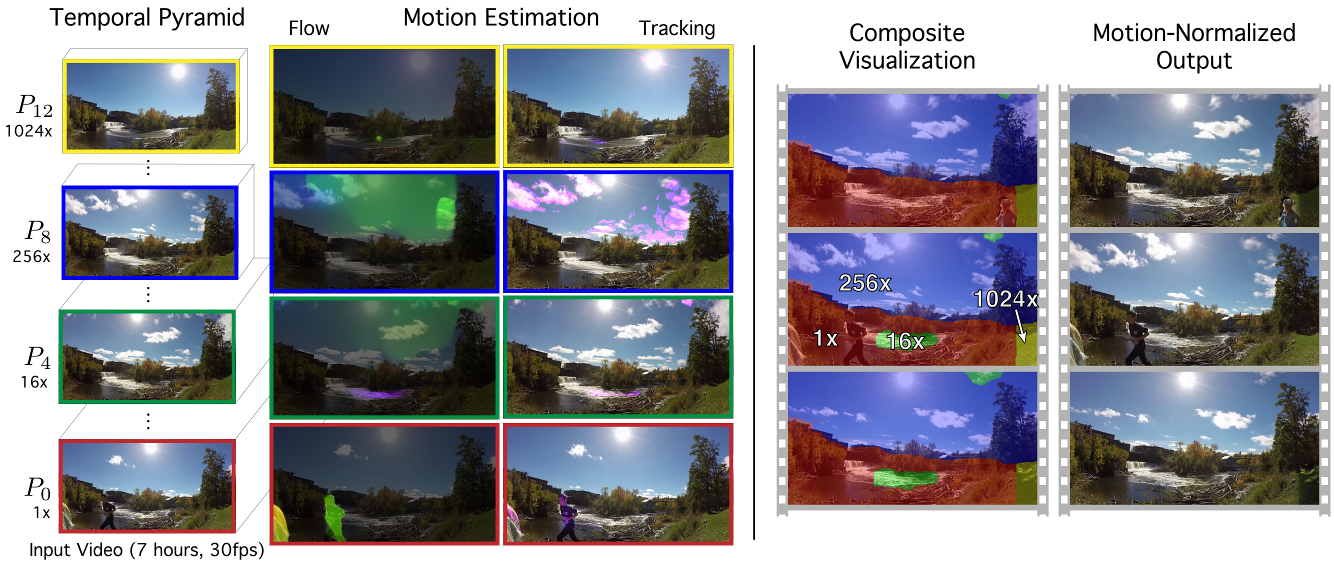


Fig. 2: Our method takes a long input video from a static camera (bottom left) and builds a temporal pyramid by repeatedly speeding it up (left column). We then estimate motion using optical flow and tracking (middle columns) to determine where motion occurs in the pyramid. Finally, we normalize the scene's motion by compositing content from multiple speeds into a seamless output video. On the left, we show one frame from each input timescale; on the right, three frames (about 50 frames apart) from the output video. The tint colors correspond to the different levels of the pyramid. See video for full results.

Recent work has explored the extremes of spatial and temporal scales, including magnification of tiny subpixel motions [3] and timelapse sequences showing years-long temporal variations [4]. In contrast, our work visualizes temporal changes over multiple timescales at once.

Temporal sampling. One simple way to manipulate time is to sample frames, either at a constant or a dynamic frame rate. [5] vary the framerate of a high-speed video according to a spatio-temporal saliency measure on the video content to show quick motions in more detail. Similarly, in the context of timelapse video, [6] use simple change-based metrics to solve for a dynamic temporal sampling that highlights or downplays changes in the scene. Both of these works begin by oversampling temporally at capture time to provide flexibility in global output sampling rate. We also sample at full-video rate (or faster), but unlike theirs, our output has a spatially-varying framerate.

Timescale analysis. Two-dimensional image pyramids are a standard tool in computer vision [7], but the idea of progressively downsampling is rarely applied to the time dimension. A version of temporal pyramids was introduced for compression and variable level-of-detail video in [8]. The video inpainting method of [9] operates in a coarse-to-fine manner on a spatio-temporal pyramid representation of a video. In contrast, in our method the temporal dimension is disjoint from the spatial dimensions: we downsample in time but not space in order to capture multiscale *temporal* phenomena that do not necessarily correspond to *spatial* scales.

A two-scale temporal decomposition was demonstrated by [10] in the context of removing jittery artifacts caused by real-time motion in timelapse videos. This work produces a time-lapse sequence that varies smoothly over time by finding a spatiotemporal displacement for each pixel within a local neighborhood. Our work can be seen as solving for a per-pixel temporal displacement field, but the offsets span large gaps in time with the goal of visualizing multiple timescales, rather than focusing on a small spatiotemporal neighborhood to visualize a single timescale smoothly.

Video compositing and temporal warping. Extensive work has been done on compositing videos, either using graph cuts (e.g., [11], [12]) or by first aligning videos (e.g., [13]). We primarily focus here on work that composites seams across time. [14], [15] create seamless video loops by choosing a start time and period for each pixel independently, and [16] scaled these techniques up to gigapixel panoramic videos. [17] turn a panning video into a static panoramic video loop, which requires shifting vertical scanlines back in time to line up with the earlier ones. Similarly, video summarization work such as [18] shifts detected moving objects of interest in time and packs them into a shorter output video.

These techniques use graph cuts to solve for an output with minimal seams or collisions. Our method uses similar machinery, but rather than labeling an output video with temporal *offsets* (shifts), we select the *framerate* (scale) for each output pixel while minimizing visual spatiotemporal seams.

[19] is closely related to our work in creating videos with spatiotemporally varying framerate. Our method differs in automatically selecting framerates based on motion content rather than relying on a user-supplied function; we also select from a discrete set of vastly disparate timescales, whereas their results show only small variations in framerate.

3 Temporal Pyramids

Multiscale pyramid decompositions such as the Gaussian and Laplacian pyramids are widely used for their ability to isolate and manipulate image content at different spatial frequencies. This is tremendously useful in tasks that require providing scale invariance, as well as in scale-specific tasks such as detail enhancement. Similarly, we propose to use a temporal pyramid decomposition of a long video to identify, analyze, and manipulate visual phenomena occurring in a scene at different temporal frequencies.

A straightforward video generalization of a Gaussian image pyramid would recursively smooth and downsample the video in all three dimensions at once. However, temporal changes manifest differently from spatial frequencies: a small, thin object consists of high spatial frequencies, but its motion may or may not be fast. Therefore, we argue that it is important to decouple time from the spatial dimensions.

Just as with spatial pyramids, there are many possible ways to build a temporal pyramid. As in the spatial domain, simply downsampling frames would result in high-frequency motions appearing aliased in longer timescales. In the temporal domain, the aliasing manifests as distracting flickering artifacts as objects appear and disappear from one frame to the next. As in the spatial domain, filtering is key to removing high-frequency motions before downsampling the video.

In this work, we consider a simple pyramid construction based on frame averaging. Beginning with an input video at full temporal resolution (e.g., 30 frames per second or 120 fps for high speed), level i of the pyramid, denoted P_i is built by applying a 1D temporal box filter of width 2^i and subsampling every 2^ith frame. Figure 2 shows an input input video at the bottom left with three levels of the pyramid above it.

We found that a box filter worked better than a Gaussian because the box filter allows objects to blur out more readily by spreading a moving object's mass evenly, rather than concentrating mass near the center of the object's trajectory as a Gaussian would. The pyramid can also be viewed as a stack of timelapse videos with different framerates. For this reason, such a pyramid is a natural choice if we seek to visualize motions happening at many different timescales.

4 Motion Normalization

Merging temporal phenomena from different timescales while still producing a natural-looking output video is difficult. We have two main objectives for our output video:

1) Include interesting phenomena from multiple timescales, each unfolding at a naturally observable rate.
2) Avoid visually jarring spatial and temporal seams.

To more concretely define the subjective concept of a "naturally observable" rate, we use pixel-space velocity as a proxy: intuitively, an object's motion is most naturally observed when it is moving across the frame neither too slowly nor too quickly. Although there is no single perfect velocity for all motion, there is a range outside which motion is either too small to notice or too large to be seen in detail. For example, the shadow of the tree in Figure 1 travels at about 0.003 pixels per frame (ppf) at 1x (i.e. real-time), meaning it takes over 11 seconds to move a distance of one single pixel.

However, we observe that if an object's motion is fully resolved in the input video, its velocity is moderate in *some* level of a temporal pyramid. Because going up a level of the pyramid speeds up time by a factor of two, an object's velocity at level i of the pyramid is 2^i times its velocity in the input video. So in level 10 of the pyramid, the shadow's velocity is 1024x faster, an easily observable 3 ppf. The clouds—also apparently stationary at real-time—move at about 5 ppf in level 8 of the pyramid. The people, on the other hand, move at about 2 ppf in the input video, so at higher pyramid levels they will first move unnaturally fast and then eventually blur out.

To simultaneously visualize the motions of shadows, clouds, and people, we propose *motion normalization*, a procedure which combines scene content from different levels of the pyramid that have *similar* screen-space velocity, and thus *different* real-world velocities. A natural approach would be to enumerate each type of motion (people, clouds, shadows) in a video, identify its "most natural" timescale, and combine all of them together into an output video. However, this is challenging because the phenomena may overlap in output space, or not completely cover the whole output in a stationary area of the scene. Instead we begin in the output domain and choose the pyramid level with the "most suitable" scene content to fill it while maintaining visual seamlessness.

4.1 Overview

We present an overview of our method here. We give further details on the motion detection and compositing step in Section 5, and extend the compositing to handle user-specified preferences in Section 6.

Capture. We begin by capturing a long video at a standard real-time framerate (e.g., 30fps). In some cases, 2-3 hours is enough to capture multiple interesting timescales; due to the difficulty of finding acceptable composites under significant (e.g., day/night) illumination changes, our longest videos are limited to about 16 hours. We demonstrate results on videos we captured with a GoPro camera, as well as videos saved from Internet streaming sources, such as YouTube Live.

Timescale Creation. Let I_0, the input video, be the real-time timescale. Timescale I_{k+1} is constructed recursively by temporally blurring and downsampling I_k. For efficiency, we use a box blur of width 2 frames, so that timescale I_k has speedup 2^k and duration $\frac{1}{2^k}$ compared to the input video.

Clip Selection. The user selects a subset of the timescales I_k, and from each timescale selects a clip of equal frame length. A typical workflow for this selection is to begin by watching a timescale with large speedup to see what phenomena appear, then fine-tuning the selected timescale to the desired viewing speed for those phenomena. Meanwhile, other, faster-paced phenomena are often visible at that timescale and suggest a choice of the next slower timescale, along with a starting point for the input clip. In this way, the user can efficiently explore the dynamic phenomena at a range of timescales, choosing particular moments to highlight in the composite. In our informal experiments, a typical clip selection session results in 2–4 input clips and takes 5–10 minutes.

Motion Detection and Compositing. Given a short clip from each of a few input timescales, our motion normalization method first detects motion in each clip, then composites moving content into a visually seamless output video. The next section goes into detail on how we pose this as a labeling problem and solve it using graph cut optimization. After compositing, we apply a further Poisson blending step to further remove artifacts caused by spatiotemporal seams. These steps are detailed in Sections 5.1–5.4.

User Input. Finally, Section 6 describes an optional step that allows a user to provide input to the compositing process, specifying additional constraints on which output pixels come from which input timescale.

5 Method

Having captured a video, built a pyramid of timescales, and selected clips from different timescales, our task is to synthesize a single output that composites dynamic content from each input clip. We pose the synthesis as a labeling problem, reminiscent of the Photomontage framework [20]. Each voxel (indexed by location and frame) in the output video takes the value of the corresponding voxel in one of the input videos. With this constraint, we only need to find a label for each output voxel specifying which input its value comes from. Posing the problem this way allows us to use graph cut optimization methods [21] to solve for a labeling that satisfies our motion normalization objective while maintaining smoothness.

Following the usual graph cuts formulation, we define an objective function on a 6-connected 3D grid graph whose nodes correspond to output video voxels. Data costs are associated with each voxel, and smoothness costs apply to the edges between neighboring voxels. In our case, the data cost encodes the desirability of choosing each voxel from each input video, which we derive from motion estimates (Section 5.1). The smoothness cost (Section 5.2) penalizes noticeable seams in the output. The individual terms of the cost function are summarized in Table 1, and the entire cost function is defined below in Equation 5.

5.1 Data Cost

Our cost function's data term, defined per input voxel, is based on optical flow and sparse tracks computed on the input clips (we also refer to these clips as "timescales", as they come from different levels of a temporal pyramid).

Flow data cost. We wish to assign a high cost to undesirable scene content—where motion is too slow or fast—and low cost to desirable scene content, where motion is moderate. To measure the speed of motion, we use the magnitude of optical flow vectors computed using [22]. We also normalize the magnitude to its equivalent at 540p to adjust for resolution differences.

To assign a cost based on flow magnitude, a simple approach would be the distance to a chosen target flow velocity. We found that a more flexible approach is to compute a clipped distance to the closest point in a reasonable range of velocities; in our experiments we used the range from 4 to 12. Formally, if the flow magnitude at an input voxel p in timescale ℓ is denoted $M(p, \ell)$, the flow-based data cost, shown graphically in Figure 3, is defined as:

$$C_{\text{flow}}(p, \ell) = \max(0, \min(8, \text{abs}(M_{p,\ell} - 8) - 4)) \tag{1}$$

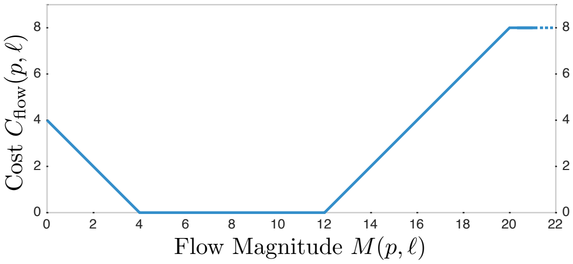


Fig. 3: C_{flow} as a function of flow magnitude

The cost of velocity zero is less than that of very high velocities because no motion is visually preferable to aliased, overly fast motion. We found this cost to work for a wide variety of phenomena. We believe this is because pixel-space velocity is a key determining factor in observability of motions, and because the sampling of timescales in our experiments is typically sparse enough that most motions fall into the low-cost range in only one timescale. Based on the phenomena being visualized and/or a user's preferences, other intervals or cost functions could be used instead.

Track data cost. Current two-frame optical flow methods applied to consecutive pairs of frames do not necessarily produce temporally smooth outputs. To help leverage information across more frames, we add a cost based on the long-term tracking algorithm of [23]. This provides a relatively sparse set of tracked points, but the robustness of the tracking is markedly better than optical flow because it is able to use more than two frames.

If a moving point is detected and tracked somewhere in an input timescale, some motion is occurring at that timescale and we want to encourage it to be included in the output. We compute tracks on each input timescale, then filter out tracks whose duration is too short and whose motion is too slow. We discard any track with a duration less than 10 frames or an average speed of less than 1 ppf (again, normalized to 540p image resolution). We could more narrowly filter tracks by velocity, but we found that in practice, points moving faster than a visually acceptable rate were not successfully tracked over enough frames to be included.

For each voxel in a given timescale containing a track, we add a penalty to all other timescales at that voxel. Let $T(p, \ell)$ be a binary indicator that is 1 if a tracked point appears at voxel p in video ℓ. The track data cost for a voxel p is:

$$C_{\text{track\_exists}}(p, \ell) = \sum_{\ell' \neq \ell} T(p, \ell') \tag{2}$$

In the overall cost function (5), this term is weighted by λ_2, which determines the strength of the penalty.

5.2 Smoothness Cost

When content from multiple timescales is naively combined, the result will generally look unnatural because the scene's appearance has changed over time; moving objects may appear or disappear, and lighting changes may create highly visible seams, either in space or time. To discourage these unnatural-looking seams, we begin with the smoothness term from [20] and augment it with an extra penalty for breaking an edge between two points in a track. We use the

Weight	Value	Term	Type	Penalize the labeling if ...
λ_1	5	$C_{\text{flow}}(p, \ell)$	unary	optical flow at voxel p is too fast or too slow in timescale ℓ
λ_2	50	$C_{\text{track\_exists}}(p, \ell)$	unary	a tracked point exists at voxel p in some other timescale $\ell' \neq \ell$
λ_3	100	$C_{\text{color\_grad}}(p, q, \ell_p, \ell_q)$	binary	colors and gradients of timescales ℓ_p and ℓ_q are visually dissimilar at p and q
λ_4	10,000	$C_{\text{track\_smooth}}(p, q, \ell_p, \ell_q)$	binary	a tracked point appears or disappears due to a change in label between p and q
λ_5	10,000	$C_{\text{user}}(p, \ell)$	unary	a user constraint has been specified at voxel p in some other timescale $\ell' \neq \ell$

TABLE 1: A summary of the terms in our cost function. We also include the weights we used in our experiments.

"Color + Gradients" variant of the Photomontage smoothness term, defined here for completeness. The cost of assigning voxels p and q labels ℓ_p and ℓ_q, respectively, is based on the RGB color and gradients of the input videos:

$$C_{\text{color\_grad}}(p, q, \ell_p, \ell_q) = C_{\text{color}} + C_{\text{grad}} \tag{3}$$

where

$$C_{\text{color}} = ||I(p, \ell_p) - I(p, \ell_q)|| + ||I(q, \ell_p) - I(q, \ell_q)||$$
$$C_{\text{grad}} = ||\nabla I(p, \ell_p) - \nabla I(p, \ell_q)|| + ||\nabla I(q, \ell_p) - \nabla I(q, \ell_q)||$$

Intuitively, if it is hard to distinguish between input ℓ_p and input ℓ_q on both sides of the seam, then it should be hard to tell that the labeling has switched from one to the other.

This smoothness term does a good job of finding the best places to switch labels, but it reasons at a very low level. To further discourage semantically jarring artifacts, such as disappearing objects, we again make use of the sparse tracking data discussed in the data term. In the context of the smoothness term, we wish to discourage the labeling from *changing* labels where a tracked object appears—a label change would correspond to the tracked object either appearing or disappearing. To discourage this, we add a new set of edges to the graph following the trajectory of each track. On these edges, the smoothness cost is the XOR of the track indicator function for the two points:

$$C_{\text{track\_smooth}}(p, q, \ell_p, \ell_q) = T(p, \ell_p) \oplus T(q, \ell_q) \tag{4}$$

In other words, a cost is paid on track-derived edges if exactly one of p and q shows the tracked object, penalizing a change of label along the trajectory of a tracked point. Once again, this binary cost is scaled by a weight λ_4.

5.3 Optimization

Putting the entire cost function together, we have

$$C(L) = \sum_p C_d(p, \ell_p) + \sum_{p,q} C_s(p, q, \ell_p, \ell_q) \tag{5}$$

where the data and smoothness costs are defined as:

$$C_d(p, \ell) = \lambda_1 C_{\text{flow}}(p, \ell) + \lambda_2 C_{\text{track\_exists}}(p, \ell) \tag{6}$$

$$\begin{aligned} C_s(p, q, \ell_p, \ell_q) = {} & \lambda_3 C_{\text{color\_grad}}(p, q, \ell_p, \ell_q) \\ & + \lambda_4 C_{\text{track\_smooth}}(p, q, \ell_p, \ell_q) \end{aligned} \tag{7}$$

The individual terms are summarized in Table 1, along with the values we used for $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. These parameters were set empirically: flow is noisy but useful where no tracks exist so its weight is small; tracks are more reliable, so they get a larger weight.

To find a labeling that approximately minimizes this objective function, we use the multi-label alpha-expansion package by Veksler and Delong.[1], which uses the min cut/max flow approach from [24], [25].

5.4 Gradient-Domain Refinement

Finally, we apply Poisson blending using the method of [26] to minimize the visual effect of the seams in the output. The aim here is the same as [20], but our implementation differs for efficiency reasons. We first create a 3-dimensional target gradient field $G(x, y, t)$—the gradient field we would like the output to satisfy, copied from the gradients of the input videos according to the labeling computed by the graph cut optimization. Then, we aim to solve the differential equation

$$G(x, y, t) = \nabla F(x, y, t) \tag{8}$$

for an output video F whose gradient matches G as closely as possible. To solve this in a least squares sense, we solve

$$\nabla^2 F = \text{div}\ G \tag{9}$$

The finite differences approximation scheme detailed by [26] produces a large sparse linear system, each row of which (for a 3-D video) corresponds to a voxel, with 6 on the diagonal and -1 in the column corresponding to each neighbor in the general case. Where some neighbors are out of bounds, the von Neumann boundary conditions imply that the diagonal is reduced to the number of in-bound neighbors.

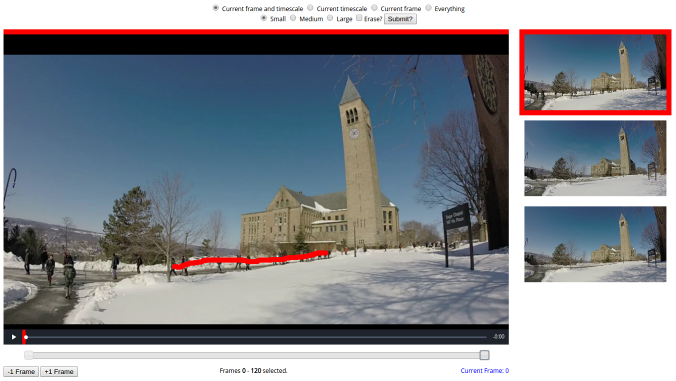
As derived, this matrix is rank-deficient, as all columns sum to 0. However, if we project this null space out of the right-hand side by subtracting the mean, we can use a conjugate gradients-style solver to arrive at a solution that is as close as possible to the desired gradient field. This is effective because CG will never change the component of the solution vector in the direction of the nullspace of the matrix [27]. We use the output from graph cuts as a good initial guess, and solve this system using stabilized biconjugate gradients as implemented in SciPy.

6 User Input

Normalizing pixel-space velocity is one reasonable heuristic for how to blend scene content from multiple timescales. In our system, a user chooses the clips of each input timescale to be included. However, a user may also desire more control over which scene content comes from which timescale.

User control is useful for several reasons. First, when multiple timescales exhibit motion at the same spatial locations, a user can choose which timescale should be displayed. For example, cloud motion often produces interesting effects at multiple timescales (see Section 7.2). Second, an artist can

[1] http://vision.csd.uwo.ca/code/ It was important to use Version 3.04 of the MAXFLOW subpackage, which includes a change that allows it to run stably on large graphs such as ours.



● Current frame and timescale ○ Current timescale ○ Current frame ○ Everything
● Small ○ Medium ○ Large □ Erase? Submit?

Frames **0** - **120** selected.

Current Frame: 0

Fig. 4: The user can scrub through each input timescale and paint annotations over a range of frames indicating extra constraints on where and when a given input timescale should be chosen to appear in the output. To create an annotation, the user selects a timescale from the thumbnails on the right, sets a range of frames, then paints the pixels that should be included from the selected frames.

achieve creative effects that do not strictly normalize scene motion. For example, the effect of speeding blurry people might illustrate the size of the crowds passing through a scene better than a short clip of the people walking in real time. Finally, the artist can correct for possible shortcomings of the motion estimation techniques used by the automatic system. The motion of flowing water is a challenging case for optical flow, so the automatic motion normalization may not choose the best timescale.

To provide user control, we built a prototype interface that allows a user to paint annotations on each timescale, indicating that the painted voxels should strongly prefer the label of the video in which they are painted. To make the process efficient, the user can specify a range of frames over which a given constraint should apply. A screenshot of the interface is shown in Figure 4, and a demonstration is provided in the video.

The constraints are then incorporated into the optimization described in Section 4 as an additional data term analogous to $C_{\text{track\_exists}}$. If $U(p, \ell')$ is a binary indicator for voxel p being painted in input ℓ, then we add the following term inside the summation of the data cost (Equation 6):

$$C_{\text{user}}(p, \ell) = \lambda_5 \sum_{\ell' \neq \ell} U(p, \ell') \tag{10}$$

7 Results

We ran our motion normalization system on eight datasets from three different sources: a GoPro camera on a tripod, live webcams, and very long YouTube videos. Notably, input data can be captured with only a tripod, an inexpensive video camera, and a spare battery pack. After capturing a video, we compute a temporal pyramid as described in Section 3. Using the pyramid to explore the activity in the video, we select equal-length input clips from a subset of the pyramid levels to normalize. The datasets, their sources, and the timescales we selected for our results are detailed in Table 2. Figure 5 shows one output frame for each dataset, and the same frame shaded to show the input source for each pixel. The timescales are labeled in order of increasing speedup as red, green, blue, and yellow. We invite the reader to view our video to see the results in full.

7.1 Automatic Motion Normalization Results

Figure 2 shows several frames from an automatic motion normalization result on the MILL dataset. The children running across the foreground are shown in real time, while parts of the flowing water are sped up. The sky is accelerated 256x to show cloud motion, and the grass on the bottom-right corner shows a shadow from the 1024x timescale towards the end of the video as the sun sets behind the trees on the right of the frame. Motion estimation is often noisy, especially at faster timescales, but the graph cuts optimization enforces a smooth output. Frames from several other fully automatic motion-normalized results are shown in the top row of Figure 5, and full clips are included in the video.

In PLAZA, the system achieves all three goals set out in Figure 1: people are shown walking at real time, while the tree shadow and clock move at a moderate pace at 1024x. The wispy clouds in the sky are not easily detected by motion estimation, but part of the cloudy sky appears at its own timescale and the blended result looks natural. BEACH depicts a variety of activities, including people, kites, waves, and clouds. The automatic result includes two timescales, a fast one capturing the moving clouds and a slower one that shows people, waves, and a large ring kite. In SLOPE, the sky and ground are assigned different timescales; the puffy clouds in the foreground drift visibly while the people walk in real time. In GEYSER, clouds move at a faster timescale while the geyser and water are shown at 1x. Some slowly rising steam in the background is shown at a more moderate speedup. Rising steam in the 64x timescale (shaded blue) poses a challenge because it is pseudo-periodic: it smooths out spatially but flickers temporally and does not blur out.

7.2 User-Guided Results

In some cases, interesting motions occur in the same region in more than one input clip. An example is SLOPE: at 256x, the puffy early afternoon clouds drift across the sky, but at 1024x the sky becomes cloudier and more textured as the sun sets. Either of these might be desired by an artist, but changing illumination makes them difficult to blend without visible artifacts. Figure 5 shows a frame in which a small block of the much faster sky appears, causing distracting motion artifacts in the video despite the single frame looking seamless. With our user interface, an artist can easily override the automatic

Dataset	Source	Length (h:m)	Timescales
ATRIUM	GoPro	8:52	8x, 64x, 4096x
BEACH	YouTube	4:03	1x, 4x, 16x, 1024x
CATER	Webcam	16:08	1x, 16x, 256x, 1024x
DAM	GoPro	(120fps) 0:41	0.25x, 1x, 64x
GEYSER	YouTube	2:50	1x, 4x, 64x, 1024x
MILL	GoPro	7:12	1x, 16x, 256x, 1024x
PLAZA	GoPro	2:45	1x, 256x, 1024x
SLOPE	GoPro	7:47	1x, 16x, 256x, 1024x

TABLE 2: List of datasets. The timescales reflect the input clips we chose for our experiments, but other choices are possible. Speedups are relative to 30fps, the output framerate.

The final version of record is available at http://dx.doi.org/10.1109/TVCG.2020.2993195

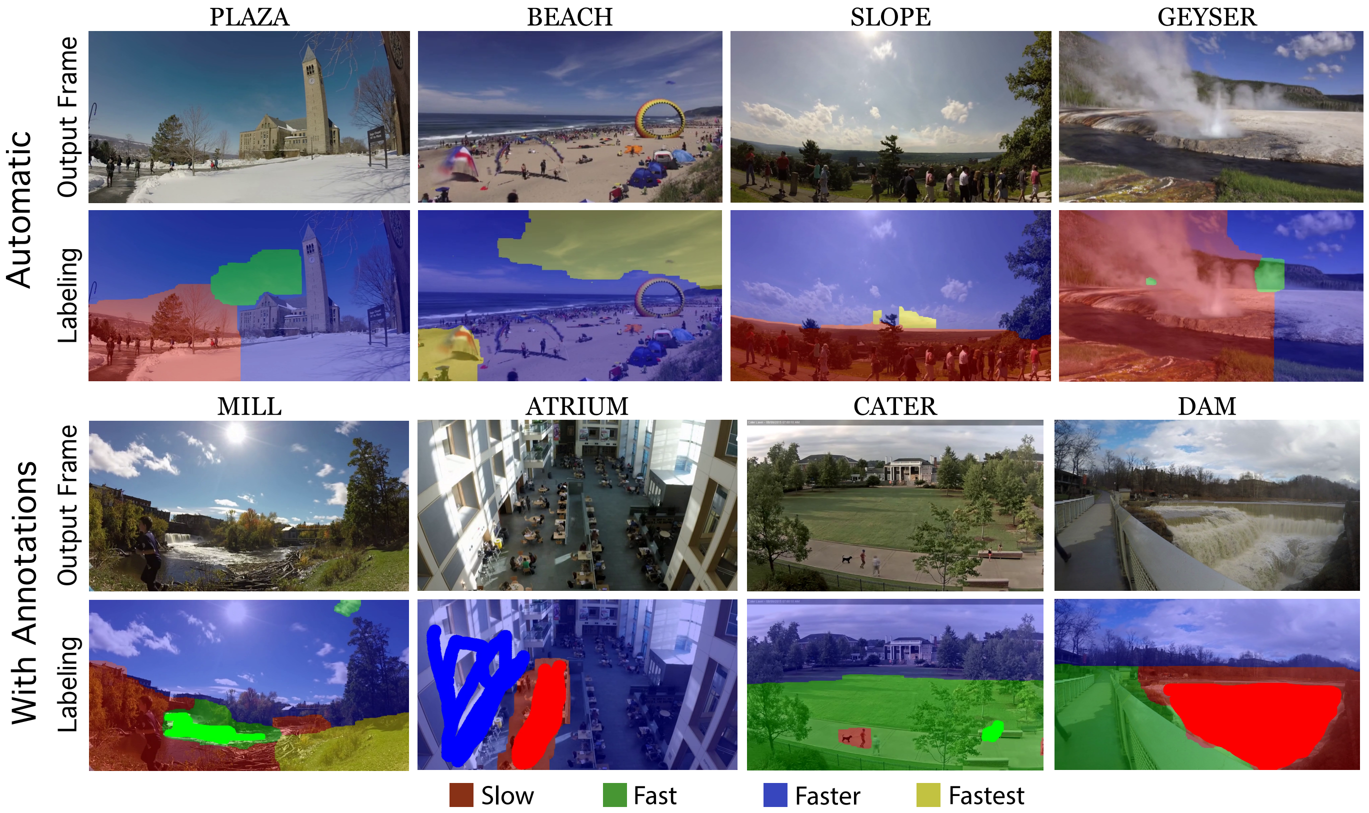


Fig. 5: Example results from our eight datasets. We show one output frame and one frame of the labeling for each result. Note that the labeling is allowed to change throughout the output; see the video for full results.

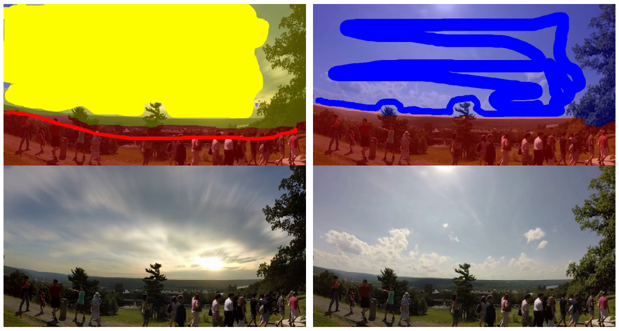


Fig. 6: Users can apply annotations to completely change which sky is chosen (left), or clean up small artifacts where skies are mixed (right; compare to the automatic result in Figure 5). In this case, the annotations were applied across the whole video, and took less than 30s to create.

system (e.g., to prefer the faster sky) or clean up artifacts due to competing motion in different timescales. Figure 6 shows two sets of annotations for SLOPE and a frame of the resulting output. With less than 30s of annotation effort the artist can swap which timescale is displayed in the sky region.

Figure 5 shows additional annotated examples. In some cases, such as CATER and DAM, the automatic result looks seamless but could show more timescales. With very little annotation effort we can increase the number of timescales depicted. We captured DAM at 120FPS, so the slowest timescale is 0.25x; with a high-framerate input, our method can simultaneously slow down fast things and speed up slow things. The result includes slow-motion water, a pedestrian on the bridge, and fast-moving clouds. In ATRIUM, people walk through the left side of the hall at 8x while afternoon sunlight sweeps across the scene at 4096x; the sharp illumination changes make for a challenging compositing problem, but using annotations yields a plausible result. The automatic method gracefully falls back to a single timescale, equivalent to a standard timelapse with frame averaging.

7.3 Additional Experiments

Pyramid Sampling. We ran our method with pyramids constructed without any temporal prefiltering; results are included as supplemental video. The method performs similarly on some datasets, but temporal aliasing artifacts throw off the motion estimation and introduce distracting jitter into some results. In MILL, some clouds in the top left of the frame briefly flicker in, creating a distracting seam in the sky. Similarly, in PLAZA the people on the right half of the walkway appear aliased.

Track Costs. To understand the importance of the unary and smoothness track costs, we experimented with leaving out the unary track cost ($\lambda_2 = 0$), the smoothness track cost ($\lambda_4 = 0$), and both ($\lambda_2 = \lambda_4 = 0$). The results are included in the supplemental video.

The unary track term has a large effect on many results due to temporally inconsistent flow estimation: the graph cut step tends to smooth over inconsistently detected motions and not include them. Tracks add temporal robustness to ensure that such consistent motions are detected and included in the output. For example, ATRIUM misses slower-scale people; BEACH is missing cloud motion in the sky; the running children in MILL are occluded by the moving water.

The smoothness track term has a much smaller effect, but does occasionally come into play. Often, the unary track term is significant enough to ensure that tracks are present; the smoothness track term helps when one tracked object attempts to occlude another. Datasets where this term makes a difference include SLOPE, where overlapping cloud motions conflict without the track smoothness term, and in ATRIUM, where the number of disparate components from slower timescales at the beginning of the video is reduced.

7.4 Runtime

The automatic result for PLAZA has 128 frames at 960x512, and takes about 4 hours on a workstation with dual 3.4GHz Intel Xeon E5 v2 CPUs and an nVidia GeForce Titan GTX. The full breakdown for PLAZA is as follows:

- Flow: 158 minutes (parallel on 32 cores)
- Tracking: 49 minutes (GPU implementation)
- Graph cuts: 129 minutes (single threaded)
- Poisson blending: 25 minutes

Importantly, flow and tracks can be precomputed for a given set of input clips. The user can then edit the compositing result using the interface, re-running only the graph cut and blending stages to create a new result. A hierarchical graph cut scheme as used in [17] could accelerate the graph cut stage, while more efficient flow, tracking, and Poisson solving algorithms could also vastly improve runtime.

7.5 Discussion and Future Work

Higher-level motion estimation techniques and/or semantic information could help resolve conflicts between motions at different timescales, such as those seen in SLOPE. More sophisticated pyramid construction techniques could detect and remove motions at each level before computing the next to avoid artifacts when quasi-periodic motions such as flags flapping (BEACH) and rising steam (GEYSER) fail to blur out at higher pyramid levels.

Dramatic illumination changes, e.g., day to night, make seamless synthesis very difficult, limiting the range of timescales that can be normalized. Future work could investigate the use of appearance transfer or illumination analysis techniques (e.g., [28], [29]), with the goal of normalizing phenomena that unfold over days, months, or years. Another direction for future work is to provide semantic-level artistic control, for example by allowing artists to specify a target velocity per object category.

8 Conclusion

We proposed a method that simultaneously visualizes motions across several timescales. We accomplish this by analyzing motion in each level of a temporal pyramid, combining content from different pyramid levels using graph cuts. We create compelling motion-normalized visualizations with spatiotemporally varying framerate, optionally incorporating artist input specified via a simple user interface. Future work has the potential to handle additional types of motion, provide higher-level control via semantic understanding, and generalize to even longer timescales.

Acknowledgments

The authors thank Dhruv Singhal and Risa Feng for help building the UI, and Shai Ben-Dor for help capturing datasets. This work was supported in part by the National Science Foundation under IIS-1111534. SW was funded by a NSF Graduate Research Fellowship under DGE-1650441.

References

[1] J. Kopf, M. F. Cohen, and R. Szeliski, "First-person hyper-lapse videos," *ACM Trans. Graph.*, 2014.

[2] N. Joshi, W. Kienzle, M. Toelle, M. Uyttendaele, and M. F. Cohen, "Real-time hyperlapse creation via optimal frame selection," *ACM Trans. Graph.*, 2015.

[3] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman, "Phase-based video motion processing," *ACM Trans. Graph.*, 2013.

[4] R. Martin-Brualla, D. Gallup, and S. M. Seitz, "Time-lapse mining from internet photos," *ACM Trans. Graph.*, 2015.

[5] F. Zhou, S. B. Kang, and M. Cohen, "Time-mapping using space-time saliency," in *IEEE CVPR*, 2014.

[6] E. P. Bennett and L. McMillan, "Computational time-lapse video," *ACM Trans. Graph.*, 2007.

[7] P. J. Burt, "Fast filter transform for image processing," *Computer Graphics and Image Processing*, 1981.

[8] A. Finkelstein, C. E. Jacobs, and D. H. Salesin, "Multiresolution video," in *SIGGRAPH*, 1996.

[9] Y. Wexler, E. Shechtman, and M. Irani, "Space-time completion of video," *IEEE TPAMI*, 2007.

[10] M. Rubinstein, C. Liu, P. Sand, F. Durand, and W. T. Freeman, "Motion denoising with application to time-lapse photography," in *IEEE CVPR*, 2011.

[11] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Trans. Graph.*, 2003.

[12] Hong-Wen Kang, Y. Matsushita, Xiaoou Tang, and Xue-Quan Chen, "Space-time video montage," in *IEEE CVPR*, 2006.

[13] Z. Cui, O. Wang, P. Tan, and J. Wang, "Time slice video synthesis by robust video alignment," *ACM Trans. Graph.*, 2017.

[14] Z. Liao, N. Joshi, and H. Hoppe, "Automated video looping with progressive dynamism," *ACM Trans. Graph.*, 2013.

[15] J. Liao, M. Finch, and H. Hoppe, "Fast computation of seamless video loops," *ACM Trans. Graph.*, 2015.

[16] M. He, J. Liao, P. V. Sander, and H. Hoppe, "Gigapixel panorama video loops," *ACM Trans. Graph.*, 2017.

[17] A. Agarwala, K. C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic video textures," in *SIGGRAPH*, 2005.

[18] Y. Pritch, A. Rav-Acha, and S. Peleg, "Nonchronological video synopsis and indexing," *IEEE TPAMI*, 2008.

[19] A. Rav-acha, Y. Pritch, D. Lischinski, and S. Peleg, "Evolving time fronts: Spatio-temporal video warping," Tech. Rep., 2005.

[20] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive digital photomontage," in *SIGGRAPH*, 2004.

[21] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE TPAMI*, 2001.

[22] M. W. Tao, J. Bai, P. Kohli, and S. Paris, "Simpleflow: A non-iterative, sublinear optical flow algorithm," *CGF (Eurographics 2012)*, 2012.

[23] N. Sundaram, T. Brox, and K. Keutzer, "Dense point trajectories by gpu-accelerated large displacement optical flow," in *ECCV*, 2010.

[24] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts," *IEEE TPAMI*, 2004.

[25] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE TPAMI*, 2004.

[26] R. Fattal, D. Lischinski, and M. Werman, "Gradient domain high dynamic range compression," *ACM Trans. Graph.*, 2002.

[27] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2013.

[28] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination for different times of day from a single outdoor photo," *SIGGRAPH Asia*, 2013.

[29] K. Sunkavalli, W. Matusik, H. Pfister, and S. Rusinkiewicz, "Factored time-lapse video," *SIGGRAPH*, 2007.