Panoramic Video Textures

Aseem Agarwala[1] Ke Colin Zheng[1] Chris Pal[3] Maneesh Agrawala[2] Michael Cohen[2]
Brian Curless[1] David Salesin[1,2] Richard Szeliski[2]

[1]University of Washington [2]Microsoft Research [3]University of Massachusetts, Amherst

Abstract

This paper describes a mostly automatic method for taking the output of a single panning video camera and creating a *panoramic video texture* (PVT): a video that has been stitched into a single, wide field of view and that appears to play continuously and indefinitely. The key problem in creating a PVT is that although only a portion of the scene has been imaged at any given time, the output must simultaneously portray motion throughout the scene. Like previous work in video textures, our method employs min-cut optimization to select fragments of video that can be stitched together both spatially and temporally. However, it differs from earlier work in that the optimization must take place over a much larger set of data. Thus, to create PVTs, we introduce a dynamic programming step, followed by a novel hierarchical min-cut optimization algorithm. We also use gradient-domain compositing to further smooth boundaries between video fragments. We demonstrate our results with an interactive viewer in which users can interactively pan and zoom on high-resolution PVTs.

Keywords: Video-based rendering, video textures, panoramas

1 Introduction

Image panoramas, in which a series of photos are taken from a single viewpoint and stitched into a single large image, date back to the mid-19th century.[1] Today, when viewed interactively on a computer with software such as QuickTime VR [Chen 1995], image panoramas offer a much more immersive experience than simple snapshots with narrower fields of view. Indeed, panoramas are now used quite commonly on the Web to provide virtual tours of hotels, museums, exotic travel destinations, and the like.

In addition to image panoramas, various hybrid image/video approaches, which include video elements playing inside of an image panorama, have been tried [Finkelstein et al. 1996; Irani and Anandan 1998]. Full video panoramas [Neumann et al. 2000; Kimber et al. 2001; Uyttendaele et al. 2004] — in which the entire scene is in motion — have also been used to convey a more visceral sense of "being there." However, such video panoramas today have two major drawbacks:

1. They require some form of specialized hardware to create, e.g., multiple synchronized video cameras [Point Grey Research 2005], or a video camera looking through a fisheye lens or pointing at a panoramically reflective mirror [Nayar 1997], which restricts the resolution of the acquired scene.[2]

Figure 1 One frame of the `waterfall` panoramic video texture.

2. They have a finite duration in time, with a specific beginning, middle, and end; this finite duration can destroy the sense of immersion.

To address the latter limitation, Schödl et al. [2000] introduced the notion of *video textures*, which are videos that appear to play continuously and indefinitely. In this paper, we describe how to create high-resolution *panoramic video textures* (PVTs), starting from just a single video camera panning across a moving scene. Specifically, this problem can be posed as follows:

> **Problem** ("PANORAMIC VIDEO TEXTURES"): Given a finite segment of video shot by a single panning camera, produce a plausible, infinitely playing video over all portions of the scene that were imaged by the camera at any time.

Here, "panning" is used to mean a camera rotation about a single axis; and "plausible" is used to mean similar in appearance to the original video, and without any visible discontinuities (or "seams"), either temporally or spatially.

The key challenge in creating a PVT is that only a portion of the full dynamic scene is imaged at any given time. Thus, in order to complete the full video panorama — *so that motion anywhere in the panorama can be viewed at any time* — we must infer those video portions that are missing. Our approach is to create a new, seamlessly loopable video that copies pixels from the original video while respecting its dynamic appearance. Of course, it is not always possible to do this successfully. While we defer a full discussion of the limitations of our algorithm to Section 6, in short, the operating range of PVTs is very similar to that of graphcut video textures [Kwatra et al. 2003]: PVTs work well for motions that are repetitive or quasi-repetitive (e.g., swaying trees) or for complex stochastic phenomena with overall stationary structure (e.g., waterfalls). PVTs do not work well for highly structured, aperiodic phenomena (e.g., the interior of a crowded restaurant).

`http://grail.cs.washington.edu/projects/panovidtex/`

[1]See, for example, the panorama of Warsaw created in 1875 at `http://www.eurofresh.se/history/1/warszawa1873-eng.htm`.

[2]The output of such a lens or mirror system is limited to the resolution of a single video camera (640×480 NTSC or 1280×720 HD), which is insufficient to create a panoramic, immersive experience that allows panning and zooming. By contrast, our panoramic video textures can reach 9 megapixels or more in size.

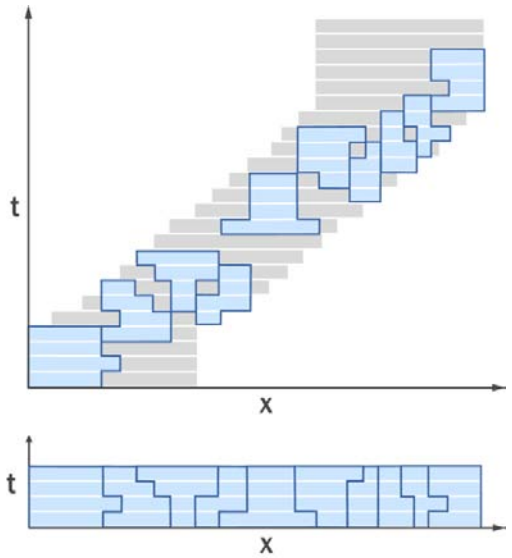

Figure 2 The top diagram shows an x,t slice of an input video volume $V(x,y,t)$. Each input video frame is shown as a grey rectangle. The frames are registered, and in this case, the camera is panning to the right. The bottom diagram shows an output video volume. The duration of the output is shorter, but each output frame is much wider than each input frame. Finally, the two diagrams show how a PVT can be constructed. The output video is mapped to locations in the input in coherent fragments; the mapping takes place in time only (as time offsets), never in space.

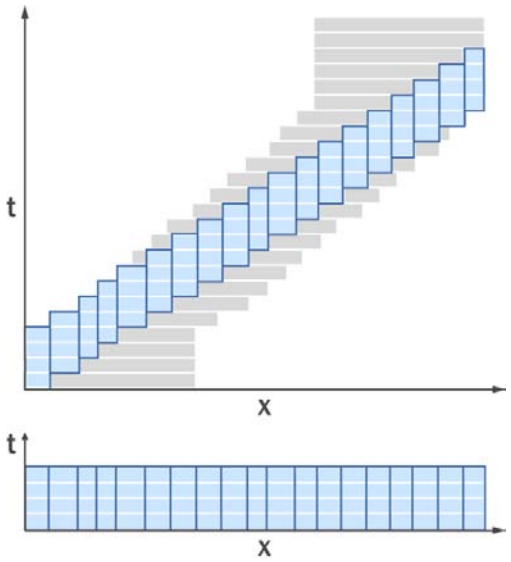

Figure 3 A simple approach to creating a PVT would be to map a continuous diagonal slice of the input video volume to the output panorama, regardless of appearance. This approach creates a valid result, but unnecessarily shears spatial structures across time (see Figure 4).

Our work builds on a series of previously published steps, including video registration, min-cut optimization, and gradient-domain compositing. Thus, the primary contributions of our work are in posing the PVT problem and in sequencing these previously published steps into a method that allows a high-resolution PVT to be created almost fully automatically from the video input of a single panning camera. (The primary input we require of the user is to segment the scene into static and dynamic portions.) A key difficulty in creating PVTs is coping with the sheer volume of high-resolution data required; previously published methods (e.g., Kwatra et al. [2003]), do not scale to problems of this size. To this end, we introduce a new dynamic programming approach, followed by a novel hierarchical min-cut optimization algorithm, which may be applicable, in its own right, to other problems. In addition, we show how gradient-domain compositing can be adapted to the creation of PVTs in order to further smooth boundaries between video fragments.

The rest of this paper is organized as follows. In the next section, we make the PVT problem more precise by defining a space for PVTs and an objective function within that space we wish to minimize. In Section 3, we introduce an optimization approach to calculating the best possible PVT within this space. In Section 4, we show how the computed result can be improved by compositing in the gradient domain. Finally, in the remaining sections of the paper, we show results, discuss limitations, and propose ideas for future research.

2 Problem definition

We begin by assuming that the input video frames are registered into a single coordinate system representing the entire spatial extent of the panorama. This registered input can be seen as forming a spatio-temporal volume $V(x,y,t)$, where x,y parameterize space (as pixel coordinates), and t parameterizes time (as a frame number from the input video).

A diagram of an x,t slice (one scanline over time) of a sample input volume is shown in Figure 2. Each input video frame provides color for only a small portion of this 3D space (the grey rectangles in Figure 2); we use the notation $V(x,y,t) = \emptyset$ to indicate that (x,y) falls outside the bounding box of the input video frame at time t, and thus has no valid color.

Our approach to constructing a PVT is to copy pixels from the input video to the output. Therefore, a PVT is represented as a mapping Δ from any pixel in the output panoramic video texture to a pixel in the input. We simplify the problem by assuming that pixels can be mapped in time but never in space. Thus, for any output pixel $p = (x,y,t)$, the mapping $\Delta(p)$ is a vector of the form $(0,0,\delta(p))$, which maps (x,y,t) to $(x,y,t+\delta(x,y,t))$. Notice that each pixel in the same piece of copied video in Figure 2 will have the same time-offset value δ.

The space of all possible PVTs is clearly quite large. One simple approach to creating a PVT might be to choose time offsets that take a sheared rectangular slice through the input volume and shear it into the output volume, as shown in Figure 3. However, such an approach may change the structure of the motion in the scene, as Figure 4 demonstrates. Also, the result is unlikely to appear seamless when played repeatedly. In contemporaneous work, Rav-Acha et al. [2005] show that this approach can sometimes be effective for creating dynamic panoramas, given the limitations noted above.

Instead, we suggest creating PVTs by mapping coherent, 3D pieces of video, as suggested in Figure 2. The seams between these pieces will be 2D surfaces embedded in the 3D space, and a good result will have visually unnoticeable seams. Therefore, we rephrase the PVT problem more precisely as follows:

> **Problem** ("PVT, TAKE 2"): Given a finite segment of video V shot by a single panning camera, create a mapping $\Delta(p) = (0,0,\delta(p))$ for every pixel p in the output panoramic video texture, such that $V(p+\Delta(p)) \neq \emptyset$ and the *seam cost* of the mapping $C_s(\Delta)$ is minimized.

It remains to define the seam cost $C_s(\Delta)$, which we will come back to after describing an additional detail. Until now we have treated

Input Simple approach Our approach

Figure 4 Three cropped details of a single frame of the `waterfall` scene (Figure 6). From left to right: the input video frame, a frame created using the simple approach described in Figure 3, and a frame created using our approach. While the simple approach yields a result that looks like a waterfall, our result more faithfully mimics the input. This comparison is shown in video form on the project web site.

PVTs as entirely dynamic; however, as the original video textures paper showed [Schödl et al. 2000], there are often important advantages to partitioning the scene into separate dynamic and static regions. For one, static regions can be stored as a single frame, saving memory. For another, spatially separate dynamic regions can be computed independently and possibly with different durations, which is useful if they portray phenomena with different periodicities.

For these reasons, we define a single static background layer $B(x,y)$. This background layer contains a color for each pixel in the user-specified static regions, while $B(x,y) = \emptyset$ for those pixels in the dynamic regions. We also define a binary matte D for the dynamic regions of the output panoramic video texture. We set D to the dilation of the null-valued regions of $B(x,y)$; thus, D overlaps the non-null regions of B along a one-pixel-wide boundary. We can use this overlap to ensure a seamless match between the dynamic and static portions of the output PVT. For convenience, we also treat D as a domain, so that $(x,y) \in D$ implies that (x,y) is in the dynamic portion of the binary matte.

With these definitions, we can define the seam cost:

$$C_s(\Delta) = \sum_{p=(x,y,t)\,|\,(x,y)\in D} (C_b(\Delta,p) + C_v(\Delta,p)) \qquad (1)$$

where

$$C_b(\Delta,p) = \begin{cases} \|V(p+\Delta(p)) - B(p)\|^k & \text{if } B(p) \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases}$$

and

$$C_v(\Delta,p) = \sum_{i=1}^{3} \begin{cases} \|V(p+\Delta(p)) - V(p+\Delta(p+e_i))\|^k & \text{if } p+e_i \in D; \\ 0 & \text{otherwise.} \end{cases}$$

Here, e_1, e_2, and e_3 are the unit vectors $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, respectively; and k is a constant, used as an exponent on the L_2 norm: we use $k = 8$. The definition of the seam cost assumes that the constraint $V(p+\Delta(p)) \neq \emptyset$ for each $p \in D$ is satisified.

Thus, the seam cost sums two terms over all pixels p in the dynamic portion of the output PVT. The first term is the difference between the pixel values in the dynamic scene and the static scene if p is on the boundary where the two overlap. The second term is the difference between the value of the pixel that p maps to in the original video and the value of the pixel that p would map to if it had the same time offset as its neighbor, for neighbors in all three cardinal directions. (In all cases, the difference is raised to the k-th power to penalize instances of higher color difference across a seam.)

One final note: in order to ensure that the PVT loops seamlessly, we define $\Delta(x,y,t) = \Delta(x,y,t \bmod t_{\max})$ for all t, where $t_{\max}$ is the duration of the PVT. We discuss how $t_{\max}$ is determined, as well as the mapping Δ itself, in the next section.

3 Our approach

Notice that the cost function C_s maps onto a 3D Markov random field (MRF), where $D \times [0..t_{\max}]$ is the domain and $\Delta(p)$ (or, equivalently, $\delta(p)$) are the free variables for which we need to solve. We can thus use optimization to minimize it. Algorithms for solving this type of problem include belief propagation [Freeman et al. 2000] and iterative min-cut [Kolmogorov and Zabih 2002].

Unfortunately, the scale of the problem is intractably large. A typical aligned input video is 6000×1200 pixels spatially, and thousands of frames long. In our experiments, the typical output video is 35 frames long, and any one pixel p has about 500 choices for $\delta(p)$. Thus, the typical output volume has 2.5×10^8 variables, each with 500 possible values. A straightforward application of standard MRF algorithms would require more memory and compute power than is currently available. We therefore designed an accelerated algorithm to compute a result.

Note that if a dynamic region is small enough to fit entirely within the bounding box of a range of $t_{\max}$ input frames, its video texture can be created using existing techniques [Kwatra et al. 2003]. For larger regions that do not meet that constraint, however, we use the following approach to integrate portions of the input video over space and time.

We first register the video frames, and then separate the PVT into static and dynamic regions. These are the only steps that require any manual user input. We then compute, for each dynamic region, a good loop length for that portion of the PVT. Finally, we solve for the mapping Δ. The first step of the solution is to minimize a heavily constrained version of the optimization problem using dynamic programming at a sub-sampled resolution. We then loosen the constraints and use min-cut optimization to refine the solution. The final step uses hierarchical min-cut optimization to refine this sub-sampled solution at the full resolution.

3.1 Video registration

Video registration is a well-studied problem, and continues to be an active area of research. We simply use existing techniques for registration (our own contributions begin after this preprocessing step). Specifically, we use a feature-based alignment method [Brown and Lowe 2003], followed by a global bundle adjustment step [Triggs et al. 2000] to align all the video frames simultaneously with each other. We use a three-parameter 3D rotation motion model [Szeliski

and Shum 1997]. Once the motion parameters are recovered, we warp the video frames into a single coordinate system by projecting them onto a cylindrical surface.

The video registration step, using existing techniques, is not always entirely successful. In order to improve the registration results, it is sometimes helpful to allow a user to manually mask out moving parts of the input video, due either to actual object motion or to scene parallax, so that they are not incorrectly used in the registration process. The masking required for the scenes we have tried is discussed in more detail in Section 5.

3.2 Static and dynamic regions

Schödl et al. [2000] showed that the dynamic regions of a scene could be identified by thresholding the variance of each pixel across time. However, we found that errors and jitter in our registration step, as well as MPEG compression noise from our digital camera, made the variance of static regions as high or higher than the variance of gently moving areas, such as rippling water. We thus ask the user to draw a single mask for each sequence. Apart from any manual masking that might be required to improve the video registration, this is the only manual part of our PVT generation process.

Given a user-drawn mask, we create a single panorama for the static regions; we first dilate the mask once to create an overlap between static and dynamic regions, and then create the static panorama automatically using the approach of Agarwala et al. [2004].

3.3 Looping length

In each dynamic region, the duration $t_{\max}$ of the output video should match the natural periodicity of that region. We thus automatically determine the natural period within a preset range $\ell_{\min} \ldots \ell_{\max}$ by examining the input video within the dynamic region. We typically set $\ell_{\min} = 30$ and $\ell_{\max} = 60$, since shorter lengths are too repetitive, and longer lengths require higher output bandwidth and compute time. To determine the best loop length $t_{\max}$, each input frame t is compared to each input frame in the range $(t+\ell_{\min}, t+\ell_{\max})$ that spatially overlaps with frame t by at least 50%. The comparison is a simple sum of squared differences in RGB values, normalized by the number of overlap pixels. We find the pair of frames $t, t+l$ that best minimizes this comparison, and set $t_{\max}(t)$ to $\ell - 1$. This approach is very simple, but in the data sets we have tried it works well.

3.4 A constrained formulation

The first step in creating a PVT for each dynamic region is to solve a heavily constrained version of the overall problem. We motivate these constraints with an observation. If a seam between spatially neighboring pixels is visually unnoticeable at a certain frame t, the seam between the same two pixels tends to be unnoticeable in the frames before and after t. Although this observation is sometimes wrong, temporal coherence in the scene makes it correct more often than not. Thus, one way to significantly reduce the search space is to assume that, at each output location (x,y), there is just a single time offset δ, regardless of t. This constraint reduces the search space from a 3D MRF to a 2D MRF. However, the 2D MRF would still be expensive to compute, since a seam penalty between neighboring output pixels would require comparing a span of pixels across time.

An additional constraint can be added to reduce the problem to a 1D Markov model. Since the video is shot by panning a video camera, we can assume that any one frame of video covers the entire height of the output space; thus, we set $\Delta(p)$ the same for each pixel in a given column of the output without creating a mapping that would go outside of the input video volume.[4]

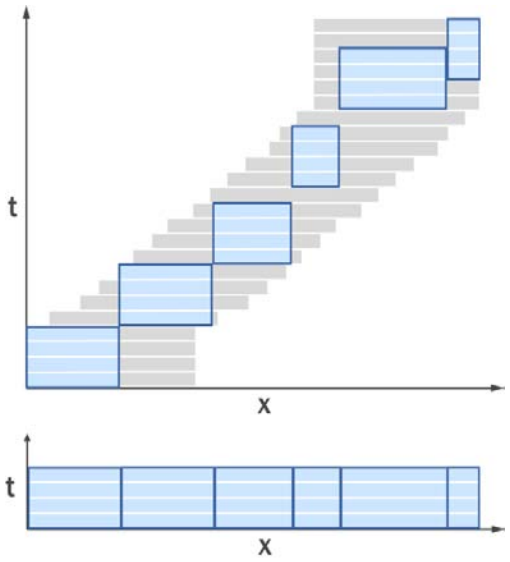

Figure 5 A possible solution to the constrained formulation described in Section 3.4, where a continuous span of video pixels across y and t is copied to each column of the output. The solution shown here uses only 6 different values of $\Delta(p)$.

Solutions meeting both these constraints have the property that $\Delta(x,y,t)$ is only a function of x (shown graphically in Figure 5). This property results in a 1D search space, requiring a choice of $\Delta(0,0,\delta(p))$ for each column of video; thus, the global minimum can be found using dynamic programming. The same cost function C_s (Equation 1) is minimized, though the C_v term will only be nonzero in the e_1 direction. The solution to the constrained formulation is a list of time offsets $\delta(0,0,\delta(p))$. We have found that this solution tends to be very coherent; that is, there are only a few unique time-offset values that end up being used, far fewer than the total number of input frames. For example, Figure 5 shows only six unique time-offsets.

3.5 Loosening the constraints

The PVT computed using the constrained formulation just described contains long, vertical cuts. These cuts may appear as artifacts, especially for more stochastic textures. Also, the computed loop is a transition between full axis-aligned blocks, like that of Schödl et al. [2000]. As Kwatra et al. [2003] showed, loops that are spatio-temporal transitions over a range of frames produce better results.

We therefore consider the full 3D MRF problem, but use the constrained solution to prune the search space. To prune the search space of the 3D MRF, we consider *only* those m unique time offsets used by the solution to the constrained formulation. In this stage of the optimization, each pixel p in the output PVT may take on a different mapping $\Delta(p) = (0,0,\delta(p))$ (Figure 2); however, $\delta(p)$ must be one of the m previously identified time offsets.

Seamless looping requires that we consider m more time offsets. Consider the case of two temporally looped neighbors $p = (x,y,0)$

[4] In practice, the full column of pixels is not always entirely present in the input volume, due to non-perfectly-horizontal panning, camera jitter, and/or the results of cylindrical mapping in the registration step. Thus, we just consider time offsets that map at least 90% of the pixels in the column to pixels in the input video volume.

and $p' = (x, y, t_{\max} - 1)$ in the output volume, which map to $(x, y, \delta(p))$ and $(x, y, t_{\max} - 1 + \delta(p'))$, respectively, in the input video. We would expect a zero seam cost if $\delta(p)$ immediately preceded $t_{\max} - 1 + \delta(p')$ in time, i.e., after rearrangement, if $\delta(p') = \delta(p) - t_{\max}$. However, the pared down list of possible time offsets may not contain $\delta(p) - t_{\max}$. Thus, to allow for a zero-cost transition in this case, we augment the set of labels to include m additional labels, $\delta(x, 0, 0) - t_{\max}$.

Given this reduced search space, iterative min-cut optimization is used to solve the more general 3D MRF. This optimization is executed in a series of *alpha expansion* moves [Kolmogorov and Zabih 2002]. Each expansion chooses a single time-offset α and expands it. During an expansion move, each pixel p can maintain its current time-offset $\delta(p)$, or switch to α. To reach a near-global minimum of the objective function, we can iterate over each possible value of α and expand it, and continue to do so until consecutively expanding each α fails to further reduce the total cost.

To perform an alpha expansion, we define a three-dimensional graph with each node representing one pixel in the output video volume, and solve for the minimum cut. Details on the graph setup can be found in Kolmogorov and Zabih [2002]. In their terminology, during a single alpha expansion the $C_b(\Delta, p)$ term of the seam cost is a function of a single binary variable, and $C_v(\Delta, p)$ is a function of two (neighboring) binary variables.

It is advantageous to minimize the size of the graph, since memory consumption and processing time scale with it. In general, it is not necessary to create a node for *each* pixel in the video volume. For example, Kolmogorov and Zabih do not create nodes for pixels already assigned time offset α, since their time offset will not change during alpha expansion. We also prune nodes in this fashion; however, our application offers another opportunity for pruning. In particular, for a given α, we do not create nodes corresponding to output pixels p for which $V(p + (0, 0, \alpha)) = \emptyset$. This pruning enforces the constraint $V(p + \Delta(p)) \neq \emptyset$ given in the PVT problem definition, and it also significantly improves efficiency. The spatial resolution of the graph becomes limited by the dimensions of an input video frame, rather than the dimensions of the entire panorama.

While pruning improves the efficiency of the optimization, we still found that performing a min-cut at full resolution can overwhelm the memory and processing capabilities of current computers. We thus present a novel, hierarchical MRF optimization approach that allows us to compute the final result.

3.6 Hierarchical min-cut optimization

A common approach for building hierarchical algorithms in computer vision [Bergen et al. 1992] is to first compute a solution at a coarser resolution. This result is then used to initialize the computation at the finer resolution, helping to avoid local minima. This approach is not immediately applicable; we cannot compute a solution at the finer resolution because the memory requirements are too high. Instead, we use the computed solution at a coarser spatial resolution *to prune* the graph when computing at the finer resolution. Our intuition is that the computed seams at the finer resolution will be roughly similar to the ones at the coarse resolution, but that the additional image information will cause local changes. We thus only optimize within the neighborhood of the seams found at the coarse resolution.

Specifically, given a result $\delta^{(k)}(x, y, t)$ computed at a coarser spatial resolution, we first create the finer resolution solution $\delta^{(k+1)}(x, y, t)$ by up-sampling $\delta^{(k)}(x, y, t)$. Then, when expanding α, we first add to the graph only those video pixels not assigned α that neighbor an α-assigned pixel. That is, we add pixels along the boundary between α and non-α. Finally, we dilate this boundary set of nodes s times (typically 10) and add these new nodes to the graph (while respecting pruning rules already mentioned). We then compute the expansion on this limited graph.

Note that this hierarchical approach is more susceptible to local minima than standard alpha-expansion. The minimum found by the alpha-expansion algorithm is provably close to the global minimum (when the energy function meets certain requirements [Kolmogorov and Zabih 2002]); our pruning technique loses this guarantee. However, in practice, in which we typically use 2 or 3 levels in the hierarchy, we have found the technique to work well.

4 Gradient-domain compositing

The result computed using optimization can still exhibit visual artifacts for several reasons. Although we lock exposure on the video camera, we still found some exposure variations in the recorded images. Also, it is sometimes impossible to find seams that are completely natural (some reasons are discussed in Section 6). Finally, small errors in the alignment procedure can also create visual artifacts. To improve results we composite the video fragments in the gradient domain, by treating the video as sources of color gradients rather than color magnitudes.

This basic approach is not new. Pérez et al. [2003] first demonstrated the efficacy of gradient-domain techniques for combining 2D images. The Photomontage system [Agarwala et al. 2004] showed that gradient-domain compositing, in combination with MRF optimization, can yield better results than either technique alone. However, simply applying this 2D approach separately to each frame of a composited video can lead to poor temporal coherence in the form of color flashing. To address this, Wang et al. [2004] extended these 2D approaches to spatio-temporal 3D in order to combine videos in the gradient domain. We build on the work of Wang et al. to solve our problem.

We first create a 2D gradient field for the still regions of the scene. This gradient field is then integrated to form the still panorama. Next, a 3D gradient field is formed for the dynamic regions of the scene, and integrated to create video textures. Integrating the gradient field requires the solution of a large, linear system of equations. A full derivation of this system can be found in Wang et al.; we describe only the unique aspects of our system. For one, we wish to ensure that seams between still and dynamic regions are minimized; this requires a judicious choice of boundary conditions. Secondly, we want to make sure the video loops seamlessly.

4.1 Boundary conditions

Any application of gradient-domain techniques first requires a choice of *boundary conditions*, which describe how to handle gradients at the boundaries of the domain. *Dirichlet* boundary conditions, like those used by Pérez et al., are suitable if the colors of the boundary pixels outside the domain are known. If they are not known, as in the Photomontage system and the work of Wang et al., the weaker *Neumann* boundary conditions must be used. The mathematical definition of these boundary conditions can be found in the cited papers.

A mix of both of these boundary conditions is needed in our case. For boundary pixels of dynamic regions that are adjacent to pixels within the still background panorama, the colors of these adjacent pixels are known; we can thus use Dirichlet boundary conditions. For boundary pixels that lie along the boundary of the entire scene, the colors of adjacent pixels outside the domain are not known; they are outside the region captured by the video camera. For these we use Neumann boundary conditions.

Figure 6 One frame of three panoramic video textures. The name, resolution, and storage requirements of the three PVTs, from top to bottom, are `waterfront` (3600x1200, 115 MB), `park` (7400x1300, 107 MB), and `yachts` (3300x800, 65 MB). The `waterfall` PVT in Figure 1 is 2600x1400, 106MB. The storage size is influenced by how much of the scene is static.

4.2 Looping

Applying the above procedure can lead to a gradual shift in color from the first to last frame; this can cause a popping artifact when looped. Thus, gradients between neighboring pixels in the first and last frame must also be added to the linear system solved when integrating the gradient field.

5 Results

We show four panoramic scenes as results. Our results are best viewed within our interactive viewer, which can be downloaded from our project web site; however, one frame of each result is shown in Figures 1 and 6. Three of these panoramas were shot with an HD video camera in portrait mode, with a vertical resolution of 1280 pixels, and one was shot with a standard video camera, with a vertical resolution of 720. We typically shot about two minutes of video for each scene, and were careful to leave the camera still at the beginning and end to capture enough material near the boundaries of the scene. (Full 360-degree PVTs should also be possible, and would not in principle require dwelling at the beginning or end.) The exposure control on each camera was locked, but auto-focus was enabled.

In general, the results are compelling and immersive, although careful examination does reveal the occasional artifact. When looking for artifacts, it is important to see if the artifact also exists in the input video. For example, there are blurry areas that arise from the limited depth of field of the camera, and noise from MPEG compression. The most noticeable artifacts, in our opinion, come from jitter and drift in the alignment. For example, jitter in a dynamic region becomes more noticeable when it is adjacent to to a perfectly still, static region.

5.1 Performance and human effort

The main human effort required to create a panoramic video texture is the drawing of a single mask; this mask does not need to be exact, and generally took us about ten minutes to create.

The automatic registration of video frames is not completely successful; the results contain jitter and drift. Scenes that lead to attractive panoramic video textures tend to contain significant motion, and this motion can be problematic for current alignment techniques. For example, the bottom twenty percent of the `waterfront` scene is filled with motion that causes significant jitter; so we cropped this region out before alignment. Techniques for registering non-rigid scenes [Fitzgibbon 2001] may reduce the need for user intervention. Finally, this same scene also contained a railing very close to the camera; since we do not rotate the camera about its exact optical center, this railing caused parallax errors. We thus cropped the bottom of the scene after alignnment, but before processing it into a panoramic video texture.

Each panoramic video texture takes between two and seven hours to compute. Significant time is spent swapping video frames in and out of memory, since the entire input video is too large to store. More intelligent caching of frames would greatly reduce the time required to produce a PVT.

It would be useful to compare the running time of our accelerated algorithm against the application of standard MRF techniques to the PVT problem (e.g., [Kwatra et al. 2003]). However, our implementation of this approach did not finish executing, since the processing of even our smallest data set would not fit in main memory. We downsampled this data set to fit within the available 2 GB of memory, but the process still did not finish after a week.

6 Limitations

We have demonstrated that high quality PVTs are possible for a variety of scenes, but they will not work for all scenes and share some of the limitations of the work of Kwatra et al. [2003]. Typically, a scene needs to exhibit some kind of stationarity, so that there exist some non-consecutive pieces of the video volume that can be shifted in time and fit together without objectionable artifacts. Examples of these kinds of scenes include phenomena with periodic or quasi-periodic motions, such as swaying trees or waving flags, and unstructured forms with more random motions, such as waterfalls.

PVTs, however, cannot model structured aperiodic phenomena, nor can they recreate periodic phenomena that were not observed for the duration of a complete cycle. Aperiodicity may arise when a single structure simply moves aperiodically, for example, when a person walks across a scene, or when overlapping elements are each moving periodically, but with sufficiently different periods that their motions do not repeat once during any given loop length. The other case, failure to observe a complete cycle of a periodic phenomenon, can arise if the camera is moved too quickly while panning.

Although PVTs cannot model structured aperiodic phenomena, in some cases, they can successfully omit them altogether. For instance, if a person walks across a scene that otherwise meets the criteria for a good PVT, the optimization algorithm will favor omitting the pixels corresponding to that person and fill in with observations from other moments in time. The input video of the `waterfall` scene, for example, contains flying birds and passing people; both are automatically omitted in the final PVT. (Another approach to handling objects such as this is to include them as “frozen” forms in the static portion of the panorama.)

Our system will sometimes produce good results for scenes that do not strictly match our assumptions. For example, the `yachts` scene contains cars on a highway in the distance. The algorithm takes advantage of occluding trees and boat masts to produce a seamless result. The distant cars seem to just disappear once they pass behind the occluders, but the overall effect is perceived as quite natural.

7 Future work

There are many good areas for future work. The ability to handle scenes with various "foreground characters" (a.k.a. "video sprites" [Schödl et al. 2000]) would greatly increase the applicability of our system. We could also eliminate the need for user interaction by automatically segmenting the scene into dynamic and still regions. This segmentation could also be used to improve the automatic registration of the video frames. Audio would greatly enhance the immersiveness of our results; the audio would require similar texturing techniques, and could follow movements within the interactive viewer. The output of multiple capture devices could be combined into one panorama. For example, the static regions could be captured with a still camera to allow greater resolution. Also, we currently require the camera to pan across the scene; it would be straightforward to loosen this constraint to allow pan and tilt, though the optimization problem would be less constrained, requiring more computation time. Finally, we currently lock the exposure on the camera. Ideally, we would let the camera choose the best exposure dynamically to create high-dynamic-range, panoramic video textures. This enhancement would involve mapping the input frames to radiance space before applying our computations.

8 Conclusion

One of the great promises of our field is to someday create a totally immersive experience, engaging all five senses in a convincing environment that could be entirely synthetic. Panoramic photography is an early technology that provides a limited form of immersion. Video textures enhance the illusion of presence, allowing dynamic phenomena to play continuously without any visual seams or obvious repetition. Panoramic video textures combine these two approaches to create what might be considered a new medium that is greater than the sum of its parts: The experience of roaming around at will in a continuous, panoramic, high-resolution, moving scene is qualitatively quite different from either panning around in a static scene or watching a single video texture play. In the not so distant future, we envision panoramic video textures being employed quite commonly on the Web, just as image panoramas are today. There, they will provide a much more compelling sense of "being there," allowing people to learn about and enjoy distant places — and share their own experiences — in captivating new ways.

Acknowledgements: This work was supported in part by a Microsoft fellowship, an industrial gift from Microsoft Research, the University of Washington Animation Research Labs, NSF grant CCR-0098005, and the Washington Research Foundation. Thanks to Drew Steedly for help with video alignment, and Wilmot Li for creating beautiful figures. Thanks to Paul Beardsley for some crucial last-minute assistance.

References

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

BERGEN, J. R., ANANDAN, P., HANNA, K. J., AND HINGORANI, R. 1992. Hierarchical model–based motion estimation. In *European Conference on Computer Vision*, 237–252.

BROWN, M., AND LOWE, D. 2003. Recognising panoramas. In *Proceedings of ICCV 03*, 1218–1225.

CHEN, S. E. 1995. Quicktime VR - an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 29–38.

FINKELSTEIN, A., JACOBS, C. E., AND SALESIN, D. H. 1996. Multiresolution video. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 281–290.

FITZGIBBON, A. W. 2001. Stochastic rigidity: Image registration for nowhere-static scenes. In *Proceedings of ICCV 2001*, 662–670.

FREEMAN, W. T., PASZTOR, E. C., AND CARMICHAEL, O. T. 2000. Learning low-level vision. *International Journal of Computer Vision 40*, 1, 25–47.

IRANI, M., AND ANANDAN, P. 1998. Video indexing based on mosaic representation. *Proceedings of IEEE 86*, 5, 905–921.

KIMBER, D., FOOTE, J., AND LERTSITHICHAI, S. 2001. Flyabout: spatially indexed panoramic video. In *Proceedings of ACM MULTIMEDIA '01*, 339–347.

KOLMOGOROV, V., AND ZABIH, R. 2002. What energy functions can be minimized via graph cuts? In *European Conference on Computer Vision (ECCV)*, 65–81.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

NAYAR, S. K. 1997. Catadioptric omnidirectional camera. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, 482.

NEUMANN, U., PINTARIC, T., AND RIZZO, A. 2000. Immersive panoramic video. In *Proceedings of MULTIMEDIA '00*, 493–494.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics 22*, 3, 313–318.

POINT GREY RESEARCH, 2005. http://ptgrey.com.

RAV-ACHA, A., PRITCH, Y., LISCHINSKI, D., AND PELEG, S. 2005. Dynamosaics: Video mosaics with non-chronological time. In *Proceedings of CVPR 2005*, To appear.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 489–498.

SZELISKI, R., AND SHUM, H.-Y. 1997. Creating full view panoramic mosaics and environment maps. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 251–258.

TRIGGS, B., MCLAUCHLAN, P. F., HARTLEY, R. I., AND FITZGIBBON, A. W. 2000. Bundle adjustment - a modern synthesis. In *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*, Springer-Verlag, 298–372.

UYTTENDAELE, M., CRIMINISI, A., KANG, S. B., WINDER, S. A. J., HARTLEY, R., , AND SZELISKI, R. 2004. High-quality image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications 24*, 3, 52–63.

WANG, H., RASKAR, R., AND AHUJA, N. 2004. Seamless video editing. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 858–861.