EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

DuctTake: Spatiotemporal Video Compositing

Jan Rüegg[1,2], Oliver Wang[2], Aljoscha Smolic[2], Markus Gross[1,2]

[1]ETH Zurich, Switzerland
[2]Disney Research Zurich, Switzerland

(a) (b) (c) (d)

Figure 1: Two examples of spatio-temporal seams (red) computed to optimally composite a pair of videos (a,c). Each seam projected onto a single frame shows complex, non-intuitive shapes that achieve high quality, temporally stable composites (b,d).

Abstract

DuctTake is a system designed to enable practical compositing of multiple takes of a scene into a single video. Current industry solutions are based around object segmentation, a hard problem that requires extensive manual input and cleanup, making compositing an expensive part of the film-making process. Our method instead composites shots together by finding optimal spatiotemporal seams using motion-compensated 3D graph cuts through the video volume. We describe in detail the required components, decisions, and new techniques that together make a usable, interactive tool for compositing HD video, paying special attention to running time and performance of each section. We validate our approach by presenting a wide variety of examples and by comparing result quality and creation time to composites made by professional artists using current state-of-the-art tools.

Categories and Subject Descriptors (according to ACM CCS): I.4.9 [Image Processing and Computer Vision]: Applications—

1. Introduction

Assembling images composed of multiple photographs is as old as photography itself. Originally achieved through arduous manual cut-and-paste, advanced digital tools now exist that make photo compositing easier (e.g., Poisson blending, alpha matting, graph cuts, etc). However, video compositing remains a challenging problem, as additional difficulties such as increased computational requirements, temporal stability, and alignment make video extensions of photographic methods non-trivial. Nonetheless, compositing video is an integral part of modern film making, and virtually all big-budget movies contain a number of scenes composed of multiple sources. Uses include special effects shots, realistic background replacement, combining optimal actor performances from multiple takes and removing unwanted scene elements or mistakes. Video compositing is still most commonly accomplished by the digital equivalent of "cut-and-paste", rotoscoping, or by chroma-keying. While chroma keying is robust and cheap, it cannot be used in all cases as it greatly restricts filming environments and often requires challenging color balancing in post production. On the other hand, rotoscoping is largely manual, expensive and time consuming and therefore is most commonly used only for expensive effect shots.

We address the problem of video compositing using a different approach. Instead of segmenting objects, we present

DuctTake (named for duct-taping together two takes), a collection of algorithms and workflows for finding optimal space-time "seams" to join together videos. Because this approach solves a simpler problem, it is robust, fast to compute, and easy for artists to use, enabling compositing techniques to be used on lower budget shots and productions. This simplification of problem domain comes at the expense of stricter prerequisites for the kinds of video clips that can be composited. Instead of compositing content from arbitrary clips, we instead focus on combining multiple *takes* together, where the content and camera location are expected to be similar; importantly, there must be enough shared scene content for low-visibility seams to exist.

The main contribution of our work is the presentation and description of an intuitive, efficient *system* and *workflow* for compositing together two FullHD video sources. This task involves difficult steps of video alignment, content matching, and motion-compensation. While individual components of this system address known problems in video processing, we believe that a detailed description of our working system will provide insight into which of the many available solutions can be practically used, especially considering strict requirements for memory usage and running time. In addition, we present several non-intuitive technical contributions, where our fast and simple methods outperformed those commonly used in the literature, including:

- Computation of spatiotemporal seams by coarse-to-fine graph cuts in a *motion-compensated* videocube.
- A robust and fast alignment technique that combines hierarchical block-based matching with drift-corrected alignment propagation.
- Efficient approximations to matching blur kernel across takes and finding optimal cropping volumes.

We validated our approach by creating a wide range of examples (available in the supplementary material along with screen-casts of their creation), and by directly comparing the quality of our results and time of creation to composites produced by professional artists using state-of-the-art tools.

2. Related Work

Our work encompasses many research areas, and builds on a large amount of prior work. Specific techniques that we used will be referenced in the appropriate implementation sections. Here we will discuss other works with goals similar to ours.

Object Segmentation Compositing is often accomplished by segmenting objects from images or videos. Numerous techniques exist where user input is leveraged to help resolve object boundaries (e.g., brush strokes [WBC*05, TZD11], rotoscoping [AHSS04], or alpha matting via trimaps [CAC*02, WAC07]). Once isolated, objects can then be pasted into desired scenes. While these methods do not require common visual elements in both videos, computing precise object boundaries is a very difficult problem. Our method instead finds seams to join similar videos. This approach has stricter content requirements, but is more robust to difficult segmentation problems such as blurred object boundaries and texture ambiguities.

Seams Probably most similar to our work is *Interactive digital photomontage* [ADA*04], which combines multiple photographs into a single output with a few simple strokes. We extend this idea to video sequences, which introduces a host of new problems such as computational tractability, temporal consistency, and alignment. Video compositing has also been addressed by computing seams in the gradient domain [WXRA07]. Gradient compositing can yield good results, but requires solving a computationally expensive integration of the gradient video volume. In addition, gradient compositing methods are highly seam dependent, and work best only with hand-chosen seams that run through largely untextured regions, as differences on the seams cause color bleeding.

Video cuts Graph cuts through video volumes have been used in the past for applications with different goals, such as automatic video-texture generation, or content extraction. *Graphcut Textures* [KSE*03] computes video textures by finding optimal seams to loop video, but did not give users the possibility to selectively choose objects from different videos. *Space-Time Video Montage* [KCMT06] performs automatic video summarization by graph cuts, and as such does not share the same goal of user-driven seamless compositing. *Panorama video textures* [AZP*05] and *Selectively De-animating Video* [BAAR12] both produce composited videos, however, they generate results with mostly static content or small looped motions. DuctTake on the other hand is a general-purpose compositing tool driven by an interactive interface. Most significantly, it is designed to operate where foreground and background can be moving arbitrarily, which introduces difficult alignment, motion compensation, and temporal stability issues.

Alignment A key component of our system is a new block-based propagate-and-refine matching technique, that is very fast, and yields temporally stable and robust results. Commonly, existing approaches align videos by feature matching [ST04]. We instead use a hierarchical compass-search, which leverages similar local search strategies widely used in the video coding community [TRRK98,ZM00]. These approaches have a fundamentally different goal; they are designed for frame-to-frame matching and need only to minimize block differences and reduce entropy [ZM00]. As a result, they have problems with aligning different views with large differences in content [TRRK98]. Our approach instead imposes a coarse-to-fine scheme which instead of merging blocks with similar motion [KL94], matches large image regions deep in the pyramid, creating smoother, spatially consistent results. This combined with a propagate-and-refine approach yields temporally consistent alignments

while still allowing for large differences in camera positions and content.

3. Algorithmic Framework

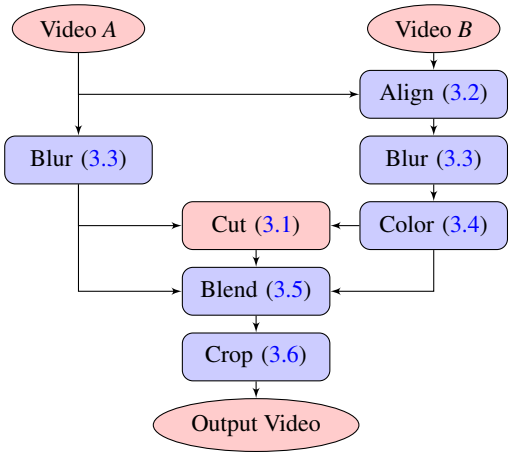


Figure 2: Algorithm overview, blue blocks are optional.

An overview of our method is presented in Figure 2 with corresponding sections in the text. The user provides a few high-level sources of information; the reference video to match colors with, the temporal offset required to synchronize events that should occur at the same time, and a few quick brush strokes indicating the parts of the video to keep from each take. The algorithm then computes an optimal seam and merges the two videos together. All performance measurements that we give throughout the paper were made with a 64bit Intel Core i7 CPU system on 1080p videos unless otherwise specified. Videos referenced in the paper are written in SMALLCAPS and can be viewed in the supplemental material under the same name.

The core technique for computing these seams is a straightforward video extension to the classic image labeling problem, for which graph cuts have long been used as a solution [BVZ01]. However, several additions are required to handle difficulties of working with video, specifically alignment due to camera motion, temporal stability, and efficient content matching, discussed in the following sections. For our application, each pixel is assigned a label that indicates which source videos it is from (A or B). A "seam" exists at locations where labels change between neighbors. Figure 1 shows two of these seams (red) that are computed using a 3D spatiotemporal graph cut (sequences CHAIR and THROUGHWINDOW). While many of the examples that we will present involve spatial cuts or temporal blends, seams can form any complex, disconnected space-time manifold.

(a) left (b) right (c) difference

Figure 3: Difference map for two frames.

3.1. Video Graph Cuts

Consider Figure 3 showing the CHAIR dataset. The goal is to composite the same person from two takes. To complicate things, the person exchanges a common object (the chair). To avoid duplication of the chair, the desired seam must pass through the object during the video. The difference map (Figure 3c), intuitively shows us similar areas, where seams will be least visible (darker regions). We introduce a seam penalty equal to the sum of the squared distance between the colors of the first and second video (A,B) at each pixel it separates. Our goal is to find a seam that separates strokes with the minimum visibility penalty.

Formally, we compute a labeling that minimizes the following visibility penalty:

$$E = \sum_i \left(\sum_{j \in N_s(i)} \delta(i,j) D(i) + \sum_{k \in N_t(i)} \lambda \delta(i,k) D(i) \right) \quad (1)$$

for all pixels i, where $D(i) = ||A_i - B_i||^2$ if A and B overlap, 0 otherwise. $N_s(i)$ are the 4 spatial and $N_t(i)$ the 2 temporal neighbors of i, and $\delta(i,j)$ is 1 if i,j are assigned different labels by the seam, else 0. The penalty for cutting temporal neighbors has different significance to cutting spatial neighbors, and is controlled by λ. A high λ penalizes the seam *moving* over a high cost area, while a low λ penalizes the seam cutting through a bad location at each frame. We show the effect of changing this parameter in Figure 4; with λ too high, the seam does not pass through the chair, which is eventually doubled in the output. $\lambda = 1$ is the default weight used in most examples.

To minimize E, we construct a graph where each node represents a pixel connected in a 3D-grid to spatial and temporal neighbors. The weights of the edges correspond to the average difference values (3c) of connected nodes. All pixels with user scribbles are connected to the source or sink respectively, and their weights set to infinity. Finally, we run a standard min-cut algorithm described by Boykov et al. [BK01] to compute the optimal labeling. We now describe two important additions to this basic strategy, motion-compensation of the video volume and a coarse-to-fine refinement, that were necessary for our application.

Motion-Compensated Seams The above graph construction assumes that there are no temporal artifacts introduced by a seam that stays in the same location from one frame to the next. However, this is only true for scenes with a static

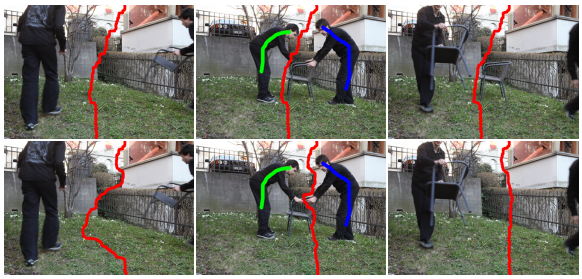
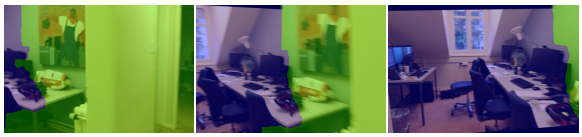


Figure 4: Three frames from the CHAIR sequence. Top, high temporal penalty. Bottom, a low temporal penalty gives more flexibility for the seam to move, allowing it to pass through the chair, which avoids doubling it in the composite.



(a) Frame 728 (b) Frame 741 (c) Frame 755

Figure 6: A temporal seam cuts from the first video (green) to the second (blue). By compensating for camera motion when computing the seam, we can hide the blend in the motion of the camera.

camera and content. When the camera or content moves, a stationary seam moves relative to the content (see Figure 5c). This creates a very disturbing effect, as differences in the background *pop up* when moving over the seam, but are not penalized in the energy formulation above (Equation 1).

We correct for this by instead constructing the graph such that nodes are connected to their location in the next frame. One way to find locations in the next frame is by optical flow. However, optical flow is expensive to compute and prone to errors, and incorrect estimates will cause artifacts in the seam movement. Instead, we model a *single* global camera motion using homography matrices, which can be robustly estimated (Section 3.2). These are then used to create a *motion-compensated* video cube, where each frame is transformed by the homography mapping it to the previous frame (as seen in Figure 5b). Then when building the compensated graph, every node in frame t is connected to the node in frame $t+1$ that corresponds to the homography transformation of that point (rounded to the nearest neighbor).

In many cases, this motion-compensation *dramatically* improved the quality of our composites. Hiding seams in camera motion that are fixed to the content is a perceptually powerful tool that in addition requires only a small region of the content to be well aligned. An example can be seen in STAIRSLOOP, and is shown in Figure 6, where we blend from one shot to the next while the camera turns into a room. Since the seam follows the content, it remains undetectable.

Coarse-to-fine Refinement Doing the above described graph cut on an HD video cube is not computationally feasible. To solve this problem we reduce the size of the graphs using a coarse-to-fine refinement approach similar to the one described in [LSGX05], but extended to 3D. In summary, we do the following:

1. Downsample the video cube (temporally *and* spatially) by a factor of two. Repeated until the resulting cube is small enough for fast processing (3 levels for 1080p video).
2. Run the graph cut on the smallest videocube
3. Grow the region around the seam by 2^{grow} pixels. A large *grow* allows more flexibility for cut locations from the previous level.
4. Upsample this region to the higher resolution cube, and add *only* pixels in the expanded seam region to the new graph.
5. Do the graph cut on the new graph.
6. Repeat from step 3, until the graph cut is run on the full resolution image.

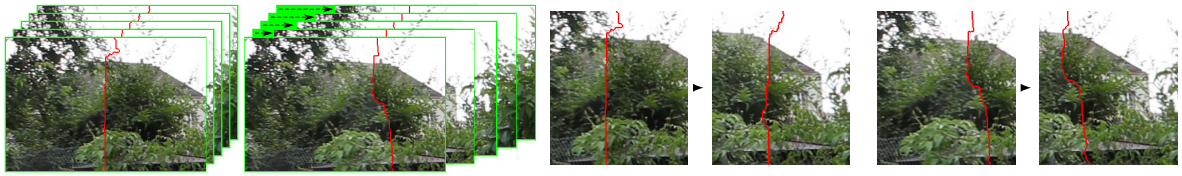
This optimization speeds up the graph cut calculation and reduces memory requirements dramatically. While actual running times are content dependent, we conducted various runtime and RAM usage experiments on an example clip of ten 1080p frames from the THIEF sequence (Figure 7). Note the huge speed and memory gains achieved by the coarse-to-fine approach. On average, using 3 pyramid levels, we compute the video seam at a rate of ~30 ms per frame.

The single scale solution computes a global optimum on the full resolution cube, taking ~5 minutes to compute and using ~5 GiB of RAM. With the typical settings we used of $level = 3$ and $grow = 1$ (used for all results shown in this work), we compute the seam in less than a second using only ~30 MiB of RAM. Furthermore, that the numerically "best" solution computed at the highest resolution can often be semantically poor (see Figure 8); in this case it cuts into the leg of the left person because of similar colors in the foreground and background. With a coarse-to-fine solution, the seam is required to be good at lower scales of the video volume as well, enforcing consistency of lower frequency content.

3.2. Alignment

Seam-based compositing works only under the condition that differences between the two takes are small in some regions. In addition, camera motion must be precisely equalized across takes, or visible wobbling and "swimming" artifacts will occur where the content has different motion. Even footage filed on a tripod requires alignment, due to small but highly visible single pixel shake. The solution (and cornerstone of our pipeline) is a robust and fast video alignment technique.

There exist many different solutions for registering/aligning two videos. After trying the most common ap-



(a) regular cube (b) motion-compensated cube (c) seam moves with frame (d) seam moves with content

Figure 5: A seam straight through the video cube stays fixed in the image, but shifts relative to the content (a,c). Compensating for motion by shifting the video cube (arrows in (b)) causes the seam to move with the content instead (b,d).

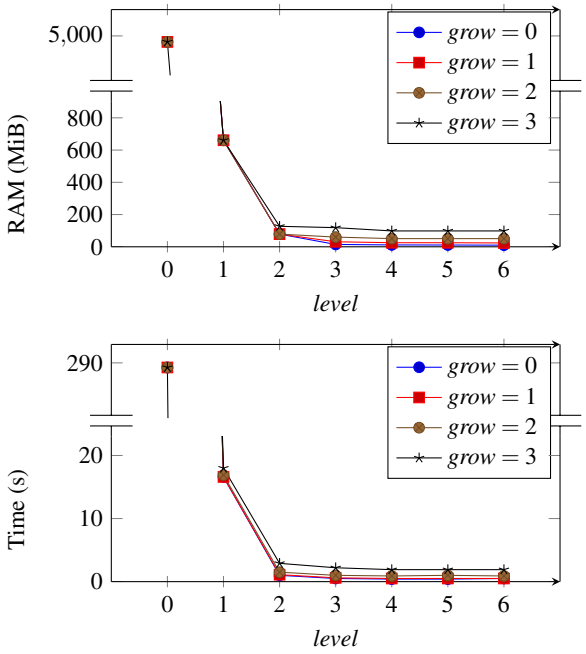
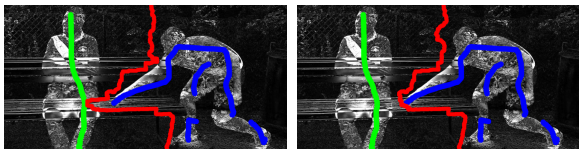


Figure 7: Time and RAM used for different settings. *level* is the number of coarse-to-fine levels (0 is the standard graph cuts approach), and 2^{grow} is of the search region in pixels added to each side of the seam before upsampling.

proaches, we found limitations in them all, and developed a simple yet robust algorithm that was significantly faster and more stable than other methods we tried. Many of these out-of-the-box techniques fail due to the large amount of camera shake and significant portions of the image with different content. CATBLURNUKE shows the result of the *F_Align* node in Nuke, a state-of-the art video compositing tool. This result took ~7 minutes to compute, compared to our method, which took ~70 seconds, and provides a more robust and stable alignment.

Alignment can be broken down into two steps; matching, where correspondences are computed between videos, and warping, where one video is warped to the view of the other. In this work, we always align the second video to the first. Which video is the "first" should be chosen by the user, as



(a) single scale (b) three levels

Figure 8: Seams (red) computed with different pyramid settings shown over difference image with user strokes (green, blue).

to maintain artistic intent, we preserve *all* camera motion in this video.

3.2.1. Matching

Optical Flow While optical flow is designed for temporal frame-to-frame matching, it can also be used to match spatially across views. After testing numerous implementations, we found state-of-the-art optical flow methods to be computationally expensive and not able to robustly handle large foreground differences and wide view separation required by our application.

Features Most commonly, images are aligned by matching features (e.g., SURF [BTVG06], or hierarchical Lucas-Kanade [Bou01]). These approaches allow robust outlier rejection and can be efficiently computed, but suffer from missing features in low-texture regions, and dramatically varying features from frame to frame. While some approaches compute more regularly spaced features by locally determined feature detection parameters [GKCE12], we observed that even slight differences in feature density from frame to frame can cause strong temporal wobbling after warping (see MIRROR example) where RANSAC (Section 3.2.2) randomly selects different planes in the world to fit to. Alternatively, while Structure From Motion (SFM) can find stable 3D points by mapping features to a single 3D model, it cannot handle scenes with low disparity or static/rotating cameras.

Block-based Block-based window-comparison methods consider all pixels evenly and therefore do not have the problem of temporally flickering features. However, these approaches are most often used for small (temporal) motion,

and fail in the case of large scene displacement. To compensate for the weaknesses of block-matching and to ensure temporal consistency, we present a coarse-to-fine propagate-and-refine approach to alignment where initial estimates are propagated temporally and then spatially refined. We first describe our fast block-matching method, called the hierarchical compass search and then discuss how we use this method to achieve a temporally stable matching.

Compass Search In its most simple form, we find a single horizontal and vertical offset dx, dy that shifts image B to match image A:

$$\arg\min_{dx,dy} \frac{\sum_{x,y} \Psi(A(x,y), B(x+dx, y+dy))}{\sum_{x,y} 1} \quad (2)$$

where Ψ is a distance function; common choices are $L1$ or $L2$ norms, or a combination gradient and color differences in different color spaces (RGB and LAB). In all examples here, we use the RGB $L1$ norm, which gave us the best ratio of quality to speed. Rather than checking all possible dx, dy (exhaustive search), we use an iterative approach. Given a dx, dy pair, we test 9 possible neighbor shifts ($dx \pm 1, dy \pm 1$), and then take the one with the minimum difference. We then perform a sub-pixel refinement by fitting a parabola through the difference at dx and its two neighbors and computing the extrema.

Of course, this search can only account for motion ≤ 1 pixel, and if iterated, it is likely to get stuck at local minima. To address both these problems, we find dx, dy using a coarse-to-fine scheme. We downsample both frames in a pyramid and start the single pixel search on the lowest level, where the pyramid height is determined by the largest expected displacement. After one step, we double dx, dy and use this as starting point for the next higher resolution level.

Hierarchical Compass Search The above approach assumes a single, rigid, translation-only model for entire frames, and cannot account for rotations or other homography effects. We therefore divide every block into some number of smaller blocks at every level that are then each independently matched. A diagram of this method can be seen in Figure 9. Displacement vectors from the various approaches are shown in Figure 10. Computation on this 720p example took ~66 ms (single-threaded) per frame, considerably less time than SURF (~1353 ms) and OpenCV's pyramid Lucas-Kanade (~236 ms), and yielded more stable, reliable results.

This method has only three, intuitive parameters: *smooth*, *level*, and *division*. *Level* controls the number of coarse-to-fine levels. *Division* controls the number of sub-blocks that each block gets divided into at the next level, which determines the final resolution of the matches. *Smooth* controls the size of the overlap region in blocks ($smooth = 1$ means an overlap of one block). We use this hierarchical compass matching to compute both an initial spatial view-to-view matching as well as a per-video temporal frame-to-frame matchings using the default parameters $level = 5, division = 5, smooth = 0$ for nearly all results. These two matching results form the basis of our temporal propagate-and-refine approach, described in detail in Section 3.2.3.

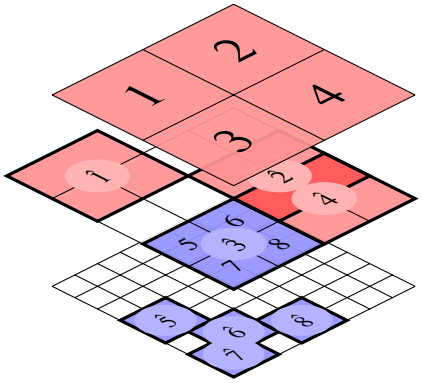


Figure 9: Hierarchical compass search example. Blocks 1-4 search for their best one-pixel displacement ($\hat{1}$-$\hat{4}$). They are then subdivided at the next level (only block 3 is shown for clarity), and again find optimal displacements ($\hat{5}$-$\hat{8}$).

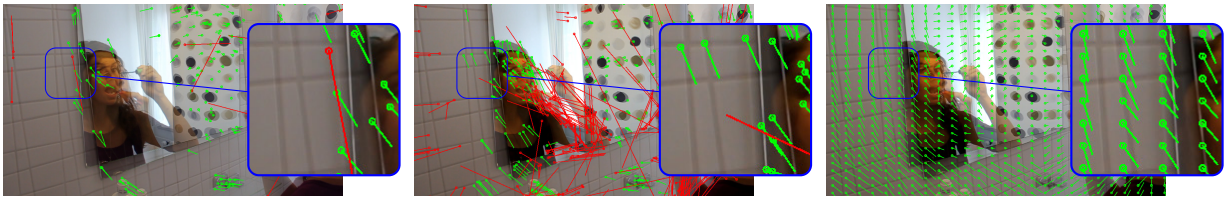
3.2.2. Warping

While each block only considers translation offsets (dx, dy), by aggregating information from multiple blocks, we can estimate more complex warping effects like rotation or perspective change. We use a homography warp (with RANSAC for outlier rejection) in all our results. However, in cases with large amounts of parallax, homographies can be insufficient to model perspective deformation. We experimented with locally varying image warps to perform alignment [LGJA09], a method commonly used in stabilization literature. However, we found that the high degrees-of-freedom of these warps cause temporal instability that creates very noticeable artifacts when warping between significantly different views (see supplemental example FUSSBALLWARP).

3.2.3. Temporal Propagation

Temporally stable alignment is especially important in our application, as we want to both synchronize the global camera motion to that of the first video, and avoid any wobbling from frame to frame. Our previously described warping can quickly match frames between views, but does not yet give temporally stable results. Simply smoothing homography matrices temporally does not work, as we do not know whether the wobbling is due to incorrect estimation or camera shake in the first video (which should be preserved).

Instead, we present a propagate-and-refine solution to address these issues using the hierarchical compass search and warping defined above. We first estimate *temporal* homography transformations in the first and second video independently. This is much easier than estimating *spatial* alignment, since the image content is very



(a) SURF (b) OpenCV (c) Hierarchical compass search

Figure 10: Example matches (red are outliers rejected by RANSAC). SURF and OpenCV are shown with empirically determined optimal parameters for this dataset.

similar, and our hierarchical compass search works without modification. We can then exploit the given temporal *and* spatial homographies (visualized as follows):

$$\begin{array}{ccc} A_{t+1} & \xleftarrow{H_{t+1}} & B_{t+1} \\ \downarrow{\scriptstyle H_a} & & \downarrow{\scriptstyle H_b} \\ A_t & \xleftarrow{H_t} & B_t \end{array}$$

where A_t and B_t are the left and right images at time t. H_a, H_b are the temporal homography warps, and H_t is the already computed spatial mapping between the two videos (the alignment) for time t. To get an estimate of H_{t+1} we can simply concatenate the given homographies in the right order:

$$H_{t+1} = H_a^{-1} * H_t * H_b \tag{3}$$

It follows that given H_a and H_b for all frames, we could compute the spatial alignment for a whole video based on one initial alignment (H_t) by essentially removing the temporal shake of the second camera, and adding the shake of the first. While this indeed reduces wobbling, it introduces drift over time because of inaccuracies in the homography estimation and numerical errors. Instead, we initialize our hierarchical compass search by transforming the second video with the propagated homography H_{t+1} before matching. Since the remaining transformation only needs to correct a very small drift, our second-pass block-based approach works very well using the default refinement parameters $level = 1, division = 4, smooth = 1$ in all datasets.

The propagation starts at any frame t where the two images are reasonably close. This is usually the frame that the user drags to align the videos. On that frame H_t is computed, and both forward and backwards H_a, H_b are used to initialize the search for H_{t+1} and H_{t-1} until all frames are aligned.

This approach forms the basis of our method, and was used in all the results shown. We allow for an optional iterative refinement step where after the initial cut is computed, the alignment is performed again, but only around the initial cut. This is used in cases where the global camera motion does not reflect the motion around the cut, for example when cutting through foreground objects (used only in sequences EYES, MIRROR and CINEMAGRAPHFUSSBALL).

(a) original composite (b) after blur matching

Figure 11: Composition before and after blur matching on sequence CATBLUR. In (a), the left half is noticeably sharper.

3.3. Blur Matching

A common problem of hand-held video footage is camera shake, which introduces motion blur. With different shake in each take, this blur can occur only in one video at a time, causing the composition to look unnatural after alignment (see Figure 11a).

De-blurring images is a challenging and under-constrained problem where fast and robust solutions are not currently available. We instead *increase* the blur of the sharper image (S) so that it matches the blurrier one (B). There are several common methods to find the local blur in images. Some calculate a spatially varying Gaussian blur using image pyramids [TMH11] while other methods estimate non-Gaussian blur kernels, for which we refer the reader to a survey paper [Rou08].

However, all of these techniques are quite slow, and for our application it is sufficient to estimate only a single global blur kernel. Therefore, we propose a simple and fast blur equalization method. We define the "blurriness" of an image to be the sum of absolute values of the gradients averaged over color channels, and computed independently for x and y dimensions. We search the filter kernel space by gradually increasing the size of the blur kernel (starting at 0) until the

(a) no matching (b) with color matching

Figure 12: Compositing with (b) and without (a) color matching.

blurriness of S equals the blurriness of B. e.g, for x:

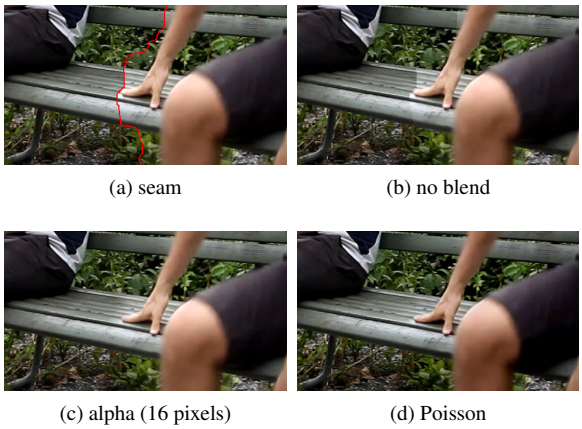
$$\arg\min_{k_x} \sum_{i \in S} ||\nabla_x F(S,k_x)||_1 - \sum_{i \in B} ||\nabla_x B||_1 \qquad (4)$$

where $F(S,k_x)$ is the result of filtering image S with kernel of radius k_x. For the choice of filter F, we use two varying-width passes of a box filter, which yielded good quality for its computational complexity. This search is done in two iterations; the first iteration searches different radii k using two passes of a width box filter, and the second iteration updates only the width of the box in the second pass. This two-step approach yields reliable convergence and fine scale refinement of kernel shapes, resulting in an optimal triangle-shaped filter. Figure 11b shows the improvement after performing blur matching on the composited result. The automatically computed two-pass box filter kernel size for this image was a 13x3 box filter followed by a 13x1 box filter. The time to compute the blur kernel depends on the amount of blur; a sequence with a large amount of motion blur (BENCHMOVING) took ~133 ms per frame to find the kernel, and ~10 ms per frame to apply it.

3.4. Color Matching

Even when filmed under similar conditions, it is likely that the two takes will differ in brightness, contrast, and hue, due to changes in lighting conditions (see Figure 12a). This is especially true for cameras that perform automatic white balance and exposure compensation, which often adapt settings while recording.

To address this, we used histogram matching independently on RGB color channels after alignment and before searching for a seam. In our application, large differences can exist in foreground content that can bias the histograms. We therefore define a similarity threshold (γ) and build the histograms only out of pixels whose difference is close enough ($||A(x,y) - B(x,y)||_1 < \gamma$).This threshold is quite robust as data is aggregated over the frames, for all results shown we set $\gamma = 200$ (for 8-bit images). It is also very efficient; calculating the lookup tables takes ~12 ms per frame, while the application of the color transform can be done in less than 3 ms. After the overlap region of the two takes ends (for example with a temporal seam), we slowly fade out the effects of color matching.



(a) seam (b) no blend

(c) alpha (16 pixels) (d) Poisson

Figure 13: Seam visibility with various blending methods on a difficult example.

3.5. Blending

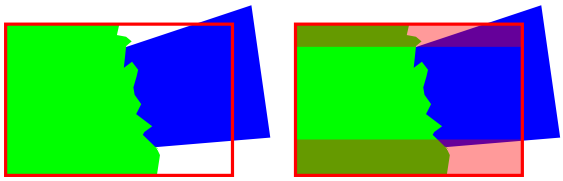
Even after color matching (Section 3.4), the seam can still be visible in a common background. This happens when there are non-global illumination differences between the images, like shadows (Figure 13b).

A fast solution is to perform simple alpha-blending around the seam. Pixels on the seam are blended equally between the two videos, and the weight falls off linearly in either direction (i.e., $\alpha A + (1-\alpha)B$). α is computed based on the Manhattan distance to the seam, and the extent of the blend can be user determined (usually between 2 to 32 pixels). This method can introduce ghosting when the backgrounds are not aligned well enough or foreground objects are close. However, since the size of the blending region is usually small, we found that this simple approach worked surprisingly well. It took on average ~33 ms per frame to compute the Manhattan distance while the actual alpha-blend took ~7 ms per frame.

For particularly difficult cases, we also added the option of a fast approximation to Poisson blending [PGB03] using convolution pyramids [FFL11]. While this method improved the result in some cases (see Figure 13d), it created well-known smearing artifacts in many others. Additionally it is much slower to compute (~345 ms per frame). Alpha blending combined with histogram matching was able to remove the effects of lighting, exposure and white balance changes in most scenes, and was used in all results presented.

3.6. Cropping

After the alignment, matching, and cut, parts of the final video can be missing (white regions in Figure 14a). To create an output video, we must crop the final video such that that no missing pixels remain in the whole sequence (if a specific aspect ratio is required, this region can be again cropped). This is an instance of the "largest empty cuboid" problem,



(a) (b)

Figure 14: A composite of two videos (green, blue) is contained inside the original video frame (red box) (a). Our greedy border shrinking approach gives crop (b).

where solutions have been shown to have a running time of $O(n^3 + n^2 \log n)$ where n is the number of pixels [NB98] in the volume. For our application, missing pixels occur largely at the edges. This observation allows us to present a greedy, iterative solution that runs in $O(ns + nt)$ time where n is the number of pixels in *one* frame, s is bounded by $\max(width, height)$, and t is the number of frames. This is very fast for video sequences, as it propagates the solution across frames, requiring only a minimal number of memory accesses. Our approach does the following:

1. Count the number of empty pixels closest to each border
2. Choose the border that has the highest number of empty pixels (rightmost border in this example)
3. Crop one pixel from this border
4. Repeat from 1 until no empty pixels are left
5. Initialize the next frame with the current crop

This approach is robust to difficult cases, such as holes and half-islands.While pathological cases can be constructed where a suboptimal cut is found, we were able to use it successfully for all results presented in this paper. Furthermore, it is very efficient in practice, requiring only ~2.5 ms per frame.

4. Implementation

Our user interface has two basic modes. In the easy mode, almost all parameters are hidden from the end user and are set to their default values. This is designed to work out-of-the-box for many tasks, the user must only load two videos, select a temporal alignment by dragging takes relative to each other and start drawing strokes on the frames. After that, he or she simply presses the *run* button and the composite is made. As needed, more strokes can be added to refine the result. For all of the optional blocks (such as blur or color matching), the user decides only whether they should be enabled or not.

The advanced mode exposes all the settings of the individual algorithms.In this form, one can experiment with other feature matchers and blend settings, tune parameters of the hierarchical compass search, RANSAC, and warping models. In general, only cases with particularly difficult camera motion and shake required using the advanced mode; the sequence STAIRSLOOP to tune alignment parameters and the sequences MIRROR and EYES for re-alignment around the cut. All the rest of the results (~90%) in this paper were generated with the easy mode, meaning that the default values for alignment and matching were used. All settings, videos, alignment, and cut data can be exported and imported in project files, and our system functions across platforms, allowing for easy collaboration.

Figure 15: Two results from our method (EMPTYSTREET and RACECAR). Left, a car is removed from the scene by compositing the same location at a different point in time. Right, two clips are combined so that a mistake is hidden.

Keyframes Users can also choose to use keyframes, which divide a single shot into multiple sections where the graph-cut functions independently. This allows working on shorter sections of long sequences, which decreases cut computation time, and guarantees that changes between two keyframes will *only* effect that region. Keyframes are realized by fixing the seam at each keyframe as a hard constraint in the graph construction. This ensures that the seam varies smoothly around keyframes, while still giving independent results in-between them. Examples shown here were composed of relatively short takes, and keyframes were not needed.

Performance High performance on FullHD video was one of the key requirements for our system. Many parts of the program exploit multi-threading, and extensive caching is done to keep the user interface responsive. The pipeline seen in Figure 2 uses a publisher-subscriber mechanism, where all intermediate data used by the steps above is cached after computation. When any of the parameters are changed by the user, the system automatically notifies all the dependent nodes and invalidates their cache, recomputing steps as needed in a lazy manner. This strategy keeps memory requirements low, while reducing the amount of recomputation required.

5. Results and Validation

Two seams are shown in Figure 15, but as this work is centered around video, we present the majority of our results in supplemental material along with timings and screen casts of their creation. Here we describe some details in reference to these datasets:

RACECAR (~4 min) This example shows DuctTake used

for correcting a filming mistake. In the first take, the car spins off of the racetrack, so a second take is made to continue on from where the mistake happened.

CINEMAGRAPH[FUSSBALL / FOUNTAIN] (~5 / ~3 min) Our method can also be used out-of-the-box to generate cinemagraphs by simply replacing one video with a static image.

PIGEONCHASE (~5 min) / CATSYNC (~4 min) Directing animals and children can be very hard. Using our compositing approach, it is possible to synchronize uncontrollable events, or to remove the trainers.

EMPTYSTREET (~6 min) This sequence shows how a single video can be used in post production to improve a shot (in this case an unwanted car is removed by taking empty street information from the same video at a temporal offset).

SPINCOMBO (~8 min) This video shows a difficult example where a temporal seam must pass through the body of a moving person. With minor iterations to correct semantic errors of the seam (like doubled balls) we achieve the final result.

BENCHMOVING (~10 min) / CHURCH (~4 min) These challenging examples show the robustness of our alignment. Despite different camera motion, strong perspective effects, and motion blur (in BENCHMOVING), our motion-compensated graph cut still finds nearly undetectable seams.

STAIRSLOOP (~12 min) / DOWNSTEPS (~5 min) These sequences show temporal seams, and are examples of how a motion-compensated cut can be used to hide a temporal transition. STAIRSLOOP contains two cuts, one for the transition into the monitor and the second to loop the video.

MIRROR (~13 min) This example shows a difficult alignment case where noticeable wobbling occurs due to large camera shake in both takes.

Comparison to Industry Tools We gave four video clips to professional digital artists with the instructions to composite the clips using state-of-the-art industry tools. We compare the resulting quality and time required to DuctTake, showing first our time and second the time taken using industry tools.

CHAIR (~2 min / ~25 min) This sequence is very easy for our system, and the method works without any of the optional blocks seen in Figure 2. Both results (ours and the artist's) look very good. Only the shadow of the chair disappears unnaturally in the artist's result.

THROUGHWINDOW (~3 min / ~30 min) In the artist's result, a noticeable jump occurs when the composition switches from the first video to the second. Because we have a much more complex temporal seam (visible in the mask) the transition is spread across multiple frames and is less visible.

EYES (~4 min / ~10 min) Both results here are good. Our method requires the additional step of re-aligning around the initial cut.

BENCHCHASE (~15 min / ~45 min) In this challenging sequence, the seam has to move very fast near the end. In the screencast one can see the iterative workflow of our method very well.

The time spent on all the remaining sequences in the supplementary material varied from 2-7 min per shot.

6. Conclusion and Outlook

In conclusion, we have presented a workflow and set of algorithms that allow a user to generate high quality composites by computing spatio-temporal seams between videos. We described the significant issues that we encountered and presented efficient solutions to these problems. Our approach is robust and intuitive, and worked over a wide range of examples, most of which were computed with fixed, default parameters. However, there are some limitations that prevent certain scenes from working well, and addressing these is an area for future work.

The most delicate component is alignment; given properly aligned views, we can almost always generate good composites with minimal work. Our use of homography warps fails when camera positions are significantly different and the desired seam must pass through objects at different depths. See FUSSBALL where wobbling is visible in the narrow left edge of the table, and INTOCAR where a jump is visible between clips. Additionally, locally varying warping approaches [LGJA09] (FUSSBALLWARP) added temporal artifacts and visible distortions. Fortunately, because our method does not require a complete frame alignment, it is often good enough to have only a small region around the seam be well aligned, for which homography warps can serve well. In addition, merely positioning cameras carefully circumvents these difficulties and greatly simplifies the alignment process.

While we focus on compositing two clips, computing a multi-way labeling instead is possible. However, the energy minimization can then only be approximated by graph cuts (e.g., via alpha expansion [BVZ01]). When this effect is desired, we can simply iteratively composite each additional video to the result one at a time (see ROOFWALK).

Using seams to composite videos can be fast and robust, but can only be used in restricted cases. For example, when desired parts from each take cross over one another, there is no similar region between objects for the seam to cut through. These types of scenes would require falling back to the existing approach of rotoscoping objects for segmentation, and pasting them on-top of each other. As future work, traditional rotoscoping could be incorporated as hard constraints into our graph construction, letting the cut handle the rest of the scene.

Finally, the numerical minimum found by graph cuts can be different from the conceptual "best" seam location, as

seams can cut through objects when their color is similar to the background. Our method uses iterative strokes to resolve these semantic problems, but interactive seam modification along the lines of panorama weaving [STP12] could be a powerful tool. Unfortunately, the optimizations used in that work to achieve interactive rates do not extend to 3D space-time seams. Further speed improvements could be made by utilizing GPU parallelism, as most steps are trivially parallelizable, which could possibly enable this kind of interactive seam modification.

Despite these limitations we believe that our system introduces a new practical paradigm for video compositing. As compared to existing state-of-the-art methods, we have shown that this system can produce comparable or better results with much less work.

Acknowledgments

We would like to thank Maurizio Nitti for generating the current state-of-the-art results to compare against the newly developed system, and a special thanks to those who appeared in our videos: Iliyan Georgiev, Antoine Milliez, Sarah Pellkofer, Simon Heinzle, Pierre Greisen, and Selina the cat.

References

[ADA*04] AGARWALA A., DONTCHEVA M., AGRAWALA M., DRUCKER S., COLBURN A., CURLESS B., SALESIN D., COHEN M.: Interactive digital photomontage. In *ACM Transactions on Graphics* (2004), vol. 23, ACM, pp. 294–302. 2

[AHSS04] AGARWALA A., HERTZMANN A., SALESIN D., SEITZ S.: Keyframe-based tracking for rotoscoping and animation. In *ACM Transactions on Graphics* (2004), vol. 23, ACM, pp. 584–591. 2

[AZP*05] AGARWALA A., ZHENG K., PAL C., AGRAWALA M., COHEN M., CURLESS B., SALESIN D., SZELISKI R.: Panoramic video textures. In *ACM Transactions on Graphics* (2005), vol. 24, ACM, pp. 821–827. 2

[BAAR12] BAI J., AGARWALA A., AGRAWALA M., RAMAMOORTHI R.: Selectively de-animating video. *ACM Transactions on Graphics 31*, 4 (2012), 66. 2

[BK01] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *Energy minimization methods in computer vision and pattern recognition* (2001), Springer, pp. 359–374. 3

[Bou01] BOUGUET J.: Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation* (2001). 5

[BTVG06] BAY H., TUYTELAARS T., VAN GOOL L.: Surf: Speeded up robust features. *Computer Vision–ECCV 2006* (2006), 404–417. 5

[BVZ01] BOYKOV Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 23*, 11 (2001), 1222–1239. 3, 10

[CAC*02] CHUANG Y., AGARWALA A., CURLESS B., SALESIN D., SZELISKI R.: Video matting of complex scenes. In *ACM Transactions on Graphics* (2002), vol. 21, ACM, pp. 243–248. 2

[FFL11] FARBMAN Z., FATTAL R., LISCHINSKI D.: Convolution pyramids. In *ACM Transactions on Graphics* (2011), vol. 30, ACM, p. 175. 8

[GKCE12] GRUNDMANN M., KWATRA V., CASTRO D., ESSA I.: Calibration-free rolling shutter removal. In *Computational Photography (ICCP), 2012 IEEE International Conference on* (2012), IEEE, pp. 1–8. 5

[KCMT06] KANG H., CHEN X., MATSUSHITA Y., TANG X.: Space-time video montage. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (2006), vol. 2, IEEE, pp. 1331–1338. 2

[KL94] KIM J., LEE S.: Hierarchical variable block size motion estimation technique for motion sequence coding. *Optical Engineering-Bellingham 33*, 8 (1994), 2553–2561. 2

[KSE*03] KWATRA V., SCHODL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3 (2003). 2

[LGJA09] LIU F., GLEICHER M., JIN H., AGARWALA A.: Content-preserving warps for 3d video stabilization. In *ACM Transactions on Graphics* (2009), vol. 28, ACM, p. 44. 6, 10

[LSGX05] LOMBAERT H., SUN Y., GRADY L., XU C.: A multilevel banded graph cuts method for fast image segmentation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 1, IEEE, pp. 259–265. 4

[NB98] NANDY S., BHATTACHARYA B.: Maximal empty cuboids among points and blocks. *Computers & Mathematics with Applications 36*, 3 (1998), 11–20. 9

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. In *ACM Transactions on Graphics* (2003), vol. 22, ACM, pp. 313–318. 8

[Rou08] ROUF M.: A study on blur kernel estimation from blurred and noisy image pairs. *CPSC548 UBC* (2008). 7

[ST04] SAND P., TELLER S.: Video matching. *ACM Transactions on Graphics 23*, 3 (2004), 592–599. 2

[STP12] SUMMA B., TIERNY J., PASCUCCI V.: Panorama weaving: fast and flexible seam processing. *ACM Transactions on Graphics 31*, 4 (2012), 83. 11

[TMH11] TRENTACOSTE M., MANTIUK R., HEIDRICH W.: Blur-aware image downsampling. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 573–582. 7

[TRRK98] THAM J., RANGANATH S., RANGANATH M., KASSIM A.: A novel unrestricted center-biased diamond search algorithm for block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on 8*, 4 (1998), 369–377. 2

[TZD11] TONG R., ZHANG Y., DING M.: Video brush: A novel interface for efficient video cutout. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 2049–2057. 2

[WAC07] WANG J., AGRAWALA M., COHEN M.: Soft scissors: an interactive tool for realtime high quality matting. In *ACM Transactions on Graphics* (2007), vol. 26, ACM, p. 9. 2

[WBC*05] WANG J., BHAT P., COLBURN R., AGRAWALA M., COHEN M.: Interactive video cutout. In *ACM Transactions on Graphics* (2005), vol. 24, ACM, pp. 585–594. 2

[WXRA07] WANG H., XU N., RASKAR R., AHUJA N.: Videoshop: A new framework for spatio-temporal video editing in gradient domain. *Graphical models 69*, 1 (2007), 57–70. 2

[ZM00] ZHU S., MA K.: A new diamond search algorithm for fast block-matching motion estimation. *Image Processing, IEEE Transactions on 9*, 2 (2000), 287–290. 2