Practical Temporal Consistency for Image-Based Graphics Applications

Manuel Lang[1,2] Oliver Wang[1] Tunc Aydin[1] Aljoscha Smolic[1] Markus Gross[1,2]
[1]Disney Research Zurich [2]ETH Zurich

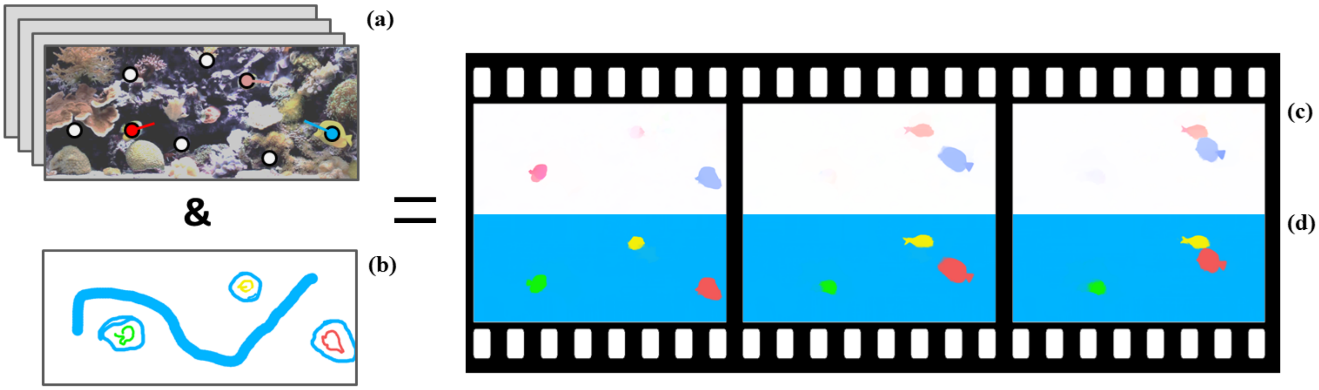

Figure 1: *One application of our method is the temporally consistent propagation of scribbles through video volumes. Sparse feature correspondences from an input video (a) are used to compute optical flow (c). Then, color scribbles (b) are spread in space and time to compute the final coherent output (d).*

Abstract

We present an efficient and simple method for introducing temporal consistency to a large class of optimization driven image-based computer graphics problems. Our method extends recent work in edge-aware filtering, approximating costly global regularization with a fast iterative joint filtering operation. Using this representation, we can achieve tremendous efficiency gains both in terms of memory requirements and running time. This enables us to process entire shots at once, taking advantage of supporting information that exists across far away frames, something that is difficult with existing approaches due to the computational burden of video data. Our method is able to filter along motion paths using an iterative approach that simultaneously uses and estimates per-pixel optical flow vectors. We demonstrate its utility by creating temporally consistent results for a number of applications including optical flow, disparity estimation, colorization, scribble propagation, sparse data up-sampling, and visual saliency computation.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Time-varying imagery;

Keywords: regularization, filtering, temporal coherence, optical flow, video

Links: DL PDF

1 Introduction

An important question for many image-based computer graphics applications is how well do these methods scale to *video sequences*. We address a general class of image-based graphics problems that solve an energy minimization combining data constraints with a regularization term enforcing smoothness (see Equation 1). Some important problems of this type are optical flow, disparity estimation, and colorization. For video applications, the temporal continuity of results is a very important requirement for preventing visible artifacts. However, many new and promising methods in this class focus only on single-frame examples, and even the benchmark Middlebury optical flow dataset (an application specifically dealing with video) provides only eight frames of video for temporal examples, and ground truth only for a single frame [Baker et al. 2011]. In contrast, the average shot length of many modern movies and TV is in the range of four to six seconds. As such, for image-based methods to be useful in video applications, they must be able to generate temporally consistent results over sequences on the order of *hundreds* of frames.

The main difficulty is that the regularization term enforcing spatial smoothness creates dependencies between pixels, often resulting in a large non-convex optimization problem. While single-frame solutions are tractable, the size of the problem increases rapidly with the inclusion of the temporal dimension, making direct extensions of these methods to video volumes computationally infeasible.

The main objective of our method is to create a memory and computationally efficient solution that enables *practical* temporal consistency for long sequences. To achieve this, we trade off accuracy for efficiency, and solve a simpler approximation of the global optimization. We use well known similarities between these optimization problems and nonlinear partial differential equations (PDEs), where anisotropic diffusion is often used to find solutions. Motivated by the close relationship between anisotropic diffusion and edge-aware filtering [Durand and Dorsey 2002], our work follows the recent trend of using image filtering techniques to efficiently solve or approximate optimization problems [Criminisi et al. 2010; Rhemann et al. 2011]. Specifically, we separate the data term from the regularization term and approximate global smoothness as an efficient local edge-aware filtering operation.

To realize such a filtering operation, we introduce an extension to the domain transform [Gastal and Oliveira 2011] that enables edge preserving filtering to be efficiently computed for video sequences while correctly following (and simultaneously computing) motion vectors in an iterative framework.

While our approach solves an approximation of the global optimization, by introducing a temporal smoothness assumption we can constrain ambiguities that would exist in a single frame and achieve high quality, temporally consistent results despite the simpler formulation. In addition, when user input must be provided (such as in the form of scribbles), our temporal continuity assumption can reduce the required manual effort by propagating this information both spatially and temporally. Unlike many recent accelerated processing methods, our approach is conceptually simple and fast without exploiting GP-GPU parallelism. In addition, it has only a few intuitive parameters, which we fixed for each application across all datasets shown in this paper.

To summarize, we present the following main contributions:

- A simple novel approach for approximating a class of energy minimization problems in image-based graphics using edge-aware filtering.
- An efficient implementation of this concept, extending existing work to enable temporal filtering that follows motion paths, confidence values, and occlusion estimates.

2 Related Work

This paper addresses a generalized method used to solve a wide array of problems, and a full review of all applications is outside the scope of the paper. Instead, we present an overview explaining how our work relates to selected relevant approaches.

Global Optimization Traditionally, image-based data + regularization problems are solved by defining a combined error term and computing a global minimum. Optical flow is commonly found by iteratively solving a linear system of equations [Horn and Schunck 1981]. Disparity estimation (stereo) methods often use a similar error formulation, which is solved with graph cuts, simulated annealing, or other such approaches [Scharstein and Szeliski 2002], and a closed form solution was presented for colorization [Levin et al. 2004]. These methods all require optimizing a large number of free variables, which leads to large memory requirements and computationally expensive convergence. Both of these problems are made worse when dealing with large (HD) images and video sequences. We present a much simpler approximation that allows us to process long, high resolution video shots.

Optical Flow Implicit computation of optical flow is an integral part of applications that produce temporally consistent results, as it is used to model the motion of objects between frames. Many high-performing modern methods still use some variation of the original Horn Schunck formulation [Horn and Schunck 1981], incorporating modified data terms and iterative, pyramid-based solutions [Sun et al. 2010; Zimmer et al. 2011]. Bilateral filtering has been integrated into an iterative variational framework, replacing the traditional anisotropic diffusion step [Xiao et al. 2006]. These methods operate only on neighboring pairs of video frames, are computationally expensive, and cannot easily enforce temporal continuity, a main focus of our work.

Other recent efficient methods filter a matching-cost-volume and select a minimum-error underlying surface to compute discretized optical flow and disparity estimates [Rhemann et al. 2011]. Like our approach, these methods also use edge-aware filtering, although for different purposes. One of their main advantages is that the optical flow is computed locally, allowing for GPGPU parallelization. However, constructing cost volumes for full video sequences is impractical given the size of the discrete label space. Our approach allows us to work in-place on the video with lower memory requirements and does not require discretization of the solution space, which can lead to artifacts.

Temporal Consistency Temporal stability has been recognized as a significant open problem. Proposed solutions include sliding windows [Hosni et al. 2011; Volz et al. 2011] and Kalman filtering [Höffken et al. 2011]. With these methods, each output frame is still computed locally, and greedy decisions can lead to temporal inconsistencies. In addition, selection of the window size is an important parameter balancing computational requirements with temporal smoothness. Our approximation avoids this trade-off as we can process entire video shots.

For colorization, methods have directly solved global optimization problems on video volumes using pre-computed optical flow to model frame-to-frame relationships [Levin et al. 2004; Bhat et al. 2010]. These approaches can generate very high quality output, but are computationally expensive and do not scale well to high resolution images or long video sequences.

Filtering Common approaches for edge aware filtering use bilateral filter weights [Tomasi and Manduchi 1998], or a local linearity assumption [He et al. 2010]. Another class of edge-aware filtering uses weights based on the geodesic-distance [Criminisi et al. 2010; Gastal and Oliveira 2011], which performs pixel mixing inversely proportional to the distance over the $\mathbb{R}^5$ $(RGBXY)$ image manifold, as opposed to the ℓ_2 norm (used for bilateral filtering). We use geodesic-distance filtering, as it more closely approximates the diffusion-model in an anisotropic heat equation, and as shown in Figure 7 yields better results for our application. Specifically, we develop an extension to the domain transform (described in further detail in Section 3).We chose this method as it is simple, efficient with large kernel sizes (necessary for our applications), and proposes a separable edge-aware filter.

In prior work, edge-aware filtering has been used for sparse data up-sampling of single frames for depth maps [Kopf et al. 2007; Yang et al. 2007], laser range data [Dolson et al. 2010], and scribbles [Gastal and Oliveira 2011; Criminisi et al. 2010]. We extend these ideas to a large class of problems by exploiting their relationship to global optimization, and tailor our approach to video volumes, efficiently addressing temporal consistency and yielding high quality results through the use of temporal smoothness.

3 Method

We address a class of problems that can be solved by minimizing error functionals of the following form,

$$E(J) = E_{data}(J) + \lambda E_{smooth}(J) \tag{1}$$

for unknown solution J, where E_{data} is the application specific error term, and E_{smooth} enforces neighborhood smoothness.

The main idea of our method is in order to avoid costly global optimization, we split up the data and smoothness terms and solve them each in series. We do this by first initializing J with application specific initial conditions that minimize E_{data} locally. The regularization term in the energy minimization (E_{smooth}) is then replaced by an efficient edge aware filtering operation on J. In this sense, smoothness is *created* as postulated, rather than solved for with an optimization. Proper formulations of this idea can be tailored to suit a variety of application scenarios which we describe in Section 4.

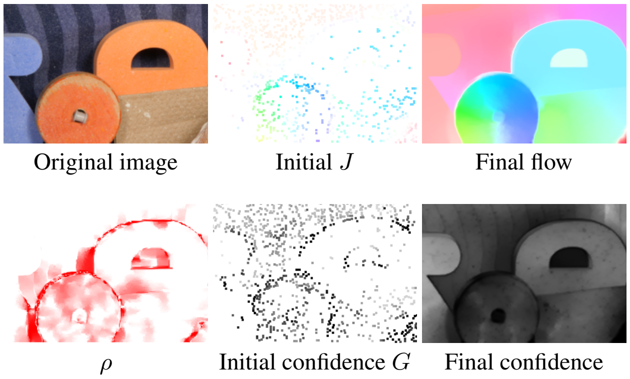

Figure 2: *Steps of our method used for solving for optical flow. We start with sparse initial estimates in J provided by feature matching (eroded for clarity). We then use our filtering approach to achieve the final result. The second row shows the occlusion weights ρ, confidence image G and the filtered confidence image.*

We will begin by explaining how our method works for a single frame, using optical flow as an example application. In this case, our unknowns $J(x, y)$ are motion vectors, expressed as $\vec{w}(x, y) = (u, v)$ corresponding to the motion between two images I_t and I_{t+1} at pixel (x, y). Equation 1 commonly becomes:

$$E_{data}(u, v) = \sum_{(x,y)} ||I_t(x + u, y + v) - I_{t+1}(x, y)||^2 \qquad (2)$$

$$E_{smooth}(u, v) = \sum_{(x,y)} (||\nabla u_{xy}||^2 + ||\nabla v_{xy}||^2) \qquad (3)$$

where ∇ is the gradient magnitude operator, E_{data} encodes the matching cost incurred by $\vec{w}$, and E_{smooth} minimizes the total quadratic variation of the flow gradient.

While replacing global optimization with a local smoothing operation has been studied before [Criminisi et al. 2010; Rhemann et al. 2011], we find it helpful to quickly go over the mathematical justification to more clearly define the class of applications that our method can be applied to. By viewing optical flow as a reaction-diffusion problem, we can see that Equation 3 is the Dirichlet's energy, and minimizing it is equivalent to solving the Laplace equation $-\Delta\vec{w} = 0$ given E_{data} as a boundary condition. This leads to the following related heat equation

$$\frac{\partial\vec{w}}{\partial t} = \alpha\Delta\vec{w} \qquad (4)$$

with the Dirac function initial conditions:

$$\vec{w}(x, y) = \begin{cases} J(x, y) & \text{if } \exists(x, y) \in J \\ \vec{0} & \text{otherwise} \end{cases} \qquad (5)$$

In the isotropic case, the Green's function solution to Equation 4 on an infinite interval is simply a Gaussian convolution. As images form an inhomogeneous medium, the arising nonlinear-PDEs can be solved by using anisotropic diffusion [Perona and Malik 1990], which in the discrete setting has been shown to be asymptotically equivalent to edge aware filtering [Paris et al. 2009]. As such, we can see that regularization equations of the form of Equation 3 can be solved with edge-aware filtering, given the right initial conditions (which are determined in the optical flow case, by Equation 2). We therefore start by initializing J with sparse feature correspondences computed between frames I_t and I_{t+1} (Equation 5). In our implementation, we use an out-of-the-box feature matcher from OpenCV that employs SIFT and Lucas-Kanade features. This provides accurate $\vec{w}$ vectors in J, but only at locations where reliable estimates can be found. We then compute an edge-aware filtering of J to create the final result (see Figure 2).

In order to realize a temporal edge-aware filtering operation, we extend recent work, called the domain transform [Gastal and Oliveira 2011], which we will briefly describe here, but refer the reader to the original work for a complete description. Rather than varying the filter weights based on image content, the signal is transformed so that it can be filtered by a fixed width Gaussian (a much more efficient, and importantly *separable* operation). It is then transformed back into the original domain, where strong gradients are maintained. Intuitively, transformed coordinates are computed for each pixel such that two pixels belonging to the same object have nearby coordinates, while pixels that lie on different sides of a strong image gradient are far apart. Figure 3 illustrates this applied to a 1D signal.

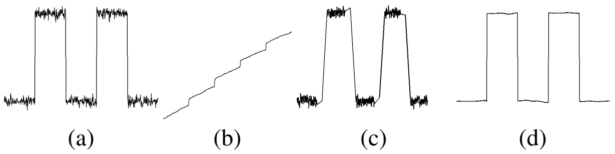

Figure 3: *1D domain transform example. (a) Input signal, (b) Transformed x-coordinates, (c) Transformed signal $\hat{C}$, (d) Output after filtering (c) with a Gaussian and re-mapping it to the original domain.*

In the 2D image case, a series of N 1D iterations is performed, alternating between X and Y until convergence. We call the dense filtered solution J'. We perform a joint domain-transform filtering, where the transformed coordinates are determined from image I, and are then used to filter the solution J.

As the Gaussian kernel is not an interpolation kernel and J may be sparse, the sum of the kernel weights at each pixel will not necessarily sum to one. A normalization image G is therefore created such that for each pixel i, $G_i = 1$ if there is data at i, and $G_i = 0$ otherwise. This image is filtered with the same filtering operation as J, producing G', which is the sum of the kernel weights per pixel.The normalized final result is then computed as $J^{''} = \frac{J'}{G'}$.

We now describe our novel extensions to this method.

Temporal Filtering We extend the single frame method to video volumes by adding an additional pass of the separable box filter that filters the temporal (T) dimension. Unlike with spatial passes, temporal filtering should follow the motion of points between frames. This prevents incorrectly averaging information across object boundaries and improves results (Figure 6). To correctly model motion, our method creates dense estimates of optical flow, regardless of the final application. We call the vector of pixels that correspond to the motion of one scene point over time a *path*. One such path is shown in Figure 4. Paths are computed by following optical flow vectors at each frame (rounding to the nearest pixel), and are then filtered after undergoing a 1D domain transform, similar to a row or column in the spatial filtering passes.

Of course, when computing optical flow, this creates a chicken-and-egg problem; per-pixel motion vectors are required in order to correctly compute the motion vectors! We resolve this using an iterative approach that begins with a rough estimate of the flow as

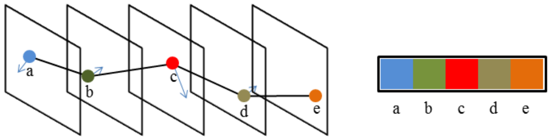

Figure 4: *A path (black line connecting pixels) is defined by the motion vectors at each current frame (blue arrows). Filtering in the temporal direction happens along these paths; the resulting 1D vector on the right is iteratively convolved with a box filter in the transformed domain.*

computed by our sparse feature matching and one spatial X and Y filter pass. This rough estimate provides initial motion vectors for the first temporal pass, which are then updated in each of the N iterations (consisting of one X, Y, T pass each).

We use a sliding box filter for efficiency, as only one addition and subtraction operation is needed for each pixel along the path, regardless of kernel size. However, we must be careful when filtering the temporal direction, as paths do not create a bijective mapping from frame to frame (which is different from rows and columns). This is due to three cases: 1) A path can leave the image boundary, 2) Multiple paths can converge on the same pixel, and 3) A pixel can have no paths in the previous frame that map to it. To compute unbiased results when filtering temporally, it is important that at any given frame, every output pixel belongs to exactly one path. To accomplish this, we begin at the first frame with one path per pixel and maintain a double-ended queue per-pixel that keeps track of the box-filter contents as it moves along that path. At each frame as we move in time and the paths begin to diverge, we find all pixels that no longer belong to a path (due to 1) or 3)). For these pixels, we spawn a new path centered at that pixel, and initialize the box filter by stepping backwards in time by the box-filter radius (in the transformed domain). In case 1), the path is ended and the sliding box filter stopped. In case 2) when multiple paths collide, we randomly keep one while cutting the other off at the previous frame. As the scale of the temporal domain is different, we use different parameters to control γ and the filter radius, which we call σ_{rt} and σ_{st} (all parameters are given in a table at the end of Section 4).

Confidence By extending the role of the normalization image G, we introduce a simple way to add confidence values to our data term. As we mentioned before, setting $G(x, y)$ to 1 in the presence of sparse data yields a normalization image that conveys how much known data has reached each pixel. To instead assign a confidence weight β_i to the sparse feature at pixel i, we set the value of $G_i = \beta_i$, and multiply the data image by the same $J_i = G_i J_i$ for all pixels i. We write this per-element multiplication as $G \cdot J$. Again, we filter both G and $G \cdot J$ and compute the final result as $\frac{(G \cdot J)'}{G'}$. In addition to being used for normalization, the confidence term correctly encourages higher confidence points to contribute a greater influence on the final result. This effect is illustrated in Figure 5.

For optical flow and disparity estimation, we use the feature matching vector difference as a confidence weight, increasing the contribution of the sparse features that had better matches.

Iterative Occlusion Estimates

We also incorporate additional information into the role of the normalization image G, which helps greatly with occlusion regions. To compute an occlusion likelihood, we estimate both forward and backwards flows ($\vec{w}^f$ and $\vec{w}^b$) at each frame, and apply a confidence penalty (ρ) for each pixel i based on how well the vectors match.

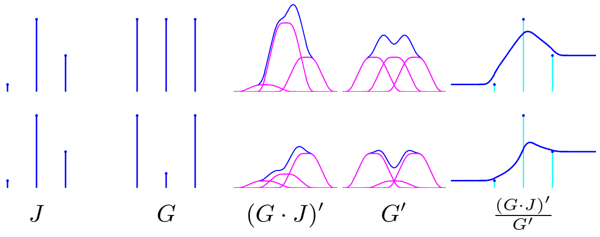

Figure 5: *A demonstration of how our confidence term is applied to two 1D examples (one each row). The original sparse signal J is modulated with the confidence G and filtered to produce $(G \cdot J)'$ (blue = signal, pink = contribution of individual points). G' is the filtered confidence image. The final normalized result $\frac{(G \cdot J)'}{G'}$ is shown with the original signal overlaid. In the bottom row, the decreased contribution of the middle point is visible due to its lower confidence value.*

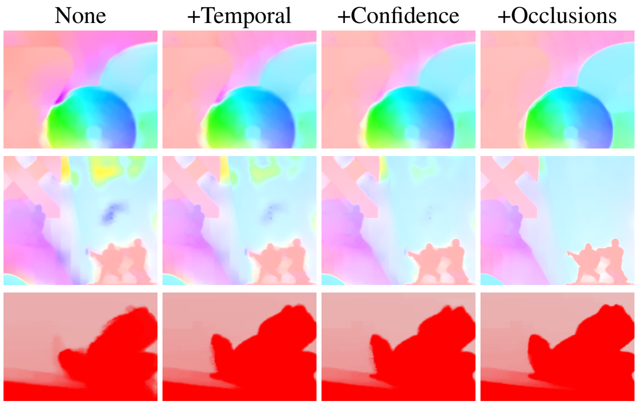

Figure 6: *Table showing the effect of each processing step introduced. We incrementally add temporal filtering, confidence values, and flow occlusion estimates to optical flow, and scribble propagation.*

We use a robust penalty function:

$$\rho = (1 - |\vec{w}^f + \vec{w}^b|)^\theta \tag{6}$$

where θ is a parameter that controls the shape of the penalty curve. While computing optical flow, these occlusion estimates ρ change each iteration (ρ^0 to ρ^{N-1}), so we update our occlusion weights based on the flow estimates from the previous iteration. Beginning with sparse data confidence $G^0 = \beta$, for each iteration n before filtering, we compute $G^n = G^{n-1} \cdot \rho^{n-1}$, and update $J^n = J^{n-1} \cdot \rho^{n-1}$ equivalently. This term has the effect of lowering the confidence in regions where our flow estimates prove unreliable. This in turn causes higher confidence spatial temporal neighbors to exert a greater influence on these occlusion regions.

Evaluation We show the improvements gained by each of the three steps in Figure 6 on several datasets, for the applications of optical flow (top two) and scribble propagation (bottom). In the first column, we show the results using the naive solution of a temporal edge aware filter (filtering the temporal dimension straight through the video volume). We then incrementally add our motion path adhering temporal filtering, confidence values, and occlusion estimates. The edge regions in particular are greatly improved, as are incorrect values that arose due to noise in our initial matching (such as on the flat background surface in the central row). In the scribble propagation example, we can see improvements in the outline of a frog. We also compare our results to other approaches for joint-data upsampling in Figure 7, showing that our method outperforms a bilateral filter [Chen et al. 2007], and a global solution computed using a locally-linear assumption [Levin et al. 2006]. The bilateral filter allows flow to spread incorrectly through similarly colored regions, and the global solution exhibits noise due to the ℓ_2 penalization of incorrect feature matches.

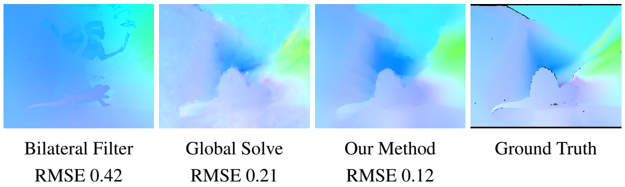

Figure 7: *A comparison of different methods for propagating sparse correspondences to compute optical flow. The results shown here were generated using each authors' publicly available code, and choosing parameters that yielded the lowest RMSE between ground truth and estimated flow vectors (in pixels).*

Quantitatively validating temporal optical flow is difficult, as public ground-truth datasets for long, real world sequences are not readily available. The widely used Middlebury ranking provides short eight-frame sequences with one frame of ground truth for testing. However, as noted by previous temporally aware methods [Hosni et al. 2011; Zimmer et al. 2011] due to capture methodology, these sequences exhibit large, temporally discontinuous motions that violate a smoothness assumption. Such prior works therefore avoided making comparisons to this data with temporal methods. Similarly, we found that our method visually performed significantly worse under these conditions than with real-world footage. However, for comparison, we include the latest Middlebury rankings; at the time of submission, our method had an average rank of 56.1 in terms of average endpoint error.

Additionally, we compare our results on real world video datasets to a number of existing state of the art methods; one that works on single frames [Zimmer et al. 2011], and two temporal approaches that use sliding windows, one variational [Volz et al. 2011] and one cost-volume filtering method [Hosni et al. 2011] (currently ranked 10th, 5th, and 15th in Middlebury evaluation respectively at the time of submission). We visualize the comparison both with color coded motion vectors, and then most significantly, by directly viewing the result of using optical flow in a typical application, in this case frame-rate upsampling. These results are shown in the supplemental video, and briefly in Figure 8, where we can see that our approach produces results that are more faithful to the movement, and less noisy temporally.

An additional advantage of our method is that by using a descriptor-based feature matcher, we are able to correctly detect small objects that exhibit large motion, as long as they contain suitable features for matching. This is something that is traditionally very difficult to model, as most variational methods linearize the image, meaning that the result is only valid locally. Long range motion is detected by computing over a scale-space pyramid, which can cause small objects to disappear.

4 Applications

We now apply our method, enforcing temporal smoothness on a number of different applications. In each of the following cases, we first compute optical flow and then compute the application-specific result, using the above described filtering approach. Figure 8 contains a table of results from our method, however we refer the viewer to the video for better validation of our method.

Disparity Estimation Disparity estimation involves computing dense correspondences between a rectified pair of stereoscopic images. For this application, solution $J_{xy} = d_{xy}$ contains scalar values that describe the disparity between a stereo image pair I^l and I^r at pixel (x, y). Similar to optical flow, this can be expressed as:

$$E_{data}(d) = \sum_{(x,y)} ||I^r(x + d_{xy}, y) - I^l(x, y)||^2$$

$$E_{smooth}(d) = \sum_{(x,y)} (||\nabla d_{xy}||^2).$$

We again sparsely compute an initial J using feature matching between I^l and I^r, and perform our filtering. We evaluate the quality of our results by comparison to high quality disparity maps provided with publicly available MPEG test sets [Wildeboer et al. 2010]. One advantage of our approach is that by design our disparities are well aligned to image edges. This is an important feature in a number of applications, such as virtual view synthesis and object insertion, which we show in the accompanying video.

Colorization and Scribble Propagation Another application of our method is the temporal and spatial propagation of sparse user input. These pen strokes can be used to colorize videos, represent high-level labeling, or to provide scene composition cues that are intuitive to humans, such as depth ordering [Wang et al. 2011]. In prior work, E_{data} enforces the given scribble map, and E_{smooth} assumes a locally-linear model [Levin et al. 2004], written for pixels i and scribbles s as:

$$E_{smooth}(s) = \sum_i ||s_i - \sum_{p \in \mathbb{N}(s_i)} w_{pi} s_p||$$

where w_{pi} are the locally linear weights. We initialize $J_i = s_i$, and apply our temporal smoothness assumption to propagate scribbles at key-frames throughout the video. For colorization, we convert the RGB image I into YCbCr space and replace the CbCr color channels with those specified by the propagated user scribbles. The required number of scribbles depends on the amount of motion in the scene, as scribbles are only valid while the scene has a similar composition. We found that in practice, we were able to get convincing results by creating on average one scribble key-frame for every 20 output frames, which is on par with sample results from other global optimization approaches.

Depth Upsampling Depth sensors often provide information that has lower spatial resolution, missing data due to parallax between sensors, and is temporally noisy. We address all of these issues by enforcing edge-aware spatial and temporal smoothness on the depth data. We tested our method by initializing J with the depth data from a Microsoft Kinect sensor, and filtering this using the video data to compute our transformed coordinates. The depth maps are shown before and after upsampling in Figure 8. We can see that our method fills in unknown areas adhering to image edges, and produces temporally consistent results for the entire sequences.

Saliency Determining visual saliency is a very important component of many image-based graphics operations. Efficient methods exist that estimate importance by analyzing the frequency spectrum [Guo et al. 2008]. We use this method to compute per-frame saliency l, initializing with $J(x, y) = l(x, y)$, and then introduce our temporal smoothness, producing a cleaned, stable output. We validate our saliency by using it for the application of video retargeting, where the aspect ratio of a video is modified while preserving the appearance of visually significant objects. In this case, the temporal stability of the saliency map is very important, as noise in the map can cause visually distracting wobbling in the final result. We also demonstrate this effect in the video using a per-frame retargeting method based on Krähenbühl et al. [2009].

We use the following parameter values for all datasets:

Application	σ_s	σ_r	σ_{st}	σ_{rt}	N	θ
Flow	2000	.4	5	.1	4	5
Disparity Estimation	2000	.3	5	.1	4	5
Scribble Propagation	1000	.3	5	.1	4	5
Depth upsampling	1000	.1	5	.1	4	5
Saliency	20	1.5	15	.2	4	5

5 Performance

An important benefit of our method is that it is computationally simple. In addition, the processing steps are mainly independent and the memory accesses mainly local, so it scales well to multi-core or hardware implementations. We provide all timing information for a HD 720p sequence on a desktop computer (Core i7 920 2.67GHz (4 cores) CPU and 12GB of memory), ignoring file IO time.

First, we load the video sequence and compute spatial transformed coordinates, which stay fixed for the sequence. This takes 3.6ms per frame.

Spatial filtering is implemented as a sliding box filter. For optical flow, six data channels per pixel (forward and backwards flow (u, v) and normalization channels G) are processed, this takes 35ms per frame. Temporal filtering also is performed with a sliding box filter, but in this case motion paths must be followed. These paths change every iteration, requiring the coordinate transform along a path to be recomputed on-the-fly. To filter a path, the sliding box filter stores a double-ended queue of entries, which avoids us from having to re-follow flow vectors as the box translates. We implemented this as a ring buffer that allows for quick push_front and pop_back operations. Additional special cases have to be handled when paths begin and end, due to the cases mentioned in Section 3. When a new paths is created, a sliding box filter is initialized around the current pixel, reconstructing the path backwards in time. This adds a small number of operations, but happens only for a few pixels in the video video (around 2.3%). While rows and columns can be trivially parallelized, filtering paths requires additional synchronization. We used an atomic check-and-set to ensure that when multiple paths converge, only one thread continues and the other ends. Temporal filtering requires a larger memory footprint to store path queues, and more random access patterns in memory. As such it is slower than the spatial passes, taking 96ms per frame.

Finally, after every XYT iteration, we update the confidence maps. This requires Equation 6 to be evaluated per pixel, which takes 20.7ms per frame.

In total, our proposed spatio-temporal filtering requires approximately 151.6ms per frame per iteration for HD video. For $N = 4$ iterations, and a shot consisting of 100 frames of 720p material, we can perform all filtering operations on six data channels (computing forward and backward flow) in 65.2 seconds; 0.652 seconds per frame.

Our method additionally requires computing sparse (forward and backward) feature correspondences. We do this for all frames in parallel, using an out-of-the-box OpenCV solution that required 145.43ms on average per frame.

We compare our method to publicly available timing information from the existing state-of-the-art optical flow methods that we validate against. Times reported are for the task of computing eight frames of optical flow on a Middlebury sequence (640x480 resolution).

Method	Time per output frame	Total for 8 frames
[Rhemann et al. 2011]	55 seconds	7.3 minutes [1]
[Zimmer et al. 2011]	620 seconds	1.4 hours
[Volz et al. 2011]	40 minutes	5.4 hours
Our method	625 ms	5 seconds

[1] We note that in the video, we compare to a more recent temporal, GPU-accelerated follow-up of this work [Hosni et al. 2011]. While its timings for optical flow have not been made public, the paper reports 41ms computing one frame of disparity estimation on a 400x300 image.

Our method does not use sliding windows, and takes only 5 seconds on a standard CPU for the 8 frames, 3.2 seconds for initial feature matching and 1.8 second for our proposed spatial-temporal filtering. Please see the video for a comparison of result quality.

Our method is also efficient in terms of memory usage, requiring order O(N) memory, where N is the number of pixels in the video volume. This allows long sequences to be computed simultaneously. Processing 400 frames of 640x480 video, or 140 frames of 1280x720 video requires 8.83 GB memory with our naive implementation. However, as most computational steps are trivially parallelizable, longer videos could easily be supported by swapping images.

6 Conclusion

In summary, we have presented a simple and efficient approach for approximating global smoothness over video sequences using temporal edge-aware filtering. Our method is robust and achieves stable results over a variety of datasets using a fixed set of parameters per application. By introducing a temporal smoothness assumption, we have shown that it is possible to obtain good results for difficult image-based computer graphics problems such as optical flow and disparity estimation by just using a feature matcher and filter in place of costly global minimization.

Our method has not been fully optimized for speed, and further performance gains could be made by exploiting GPU parallelism. Recent work describes significant performance gains computing summed area tables on the GPU, which is a large component of our computation [Nehab et al. 2011].

Of course, our approach is not without limitations. Primarily, we have decided to trade accuracy for computational efficiency, in the process greatly simplifying the problem that we are solving. One consequence of this is our dependence on having sufficient initial conditions; it is possible that objects can remain undetected when there are not enough image features to match between frames. To alleviate this, we tuned the parameters of our feature matcher to find *as many* features as possible, even when this creates a number of bad matches, as our filtering approach suffers more from the lack of data than from the presence of outliers (incorrect matches), which are largely filtered out by spatial and temporal neighbors.

Our method also shares similar limitations to most image-based computer graphics approaches in that it can fail when important object boundaries are not well represented by image edges. However, in applications with very sparse input such as scribble propagation, greedy mistakes can cause data to incorrectly bleed across boundaries, something that would be avoided in a true global solution. As a result, when compared to prior scribble propagation work, our method requires scribbles to be more localized at object boundaries; this is a trade-off for our efficiency gains. This effect can be seen in the supplementary video, where we perform colorization using scribbles from a prior global approach [Levin et al. 2004]), and show a comparison between our result and theirs. One possible solution could be to user interaction where the more important object edges are highlighted and the confusing texture edges are suppressed. As our method can provide interactive rates for single key-frames and fast results for video sequences, this kind of interaction could greatly improve the results with minimal overhead.

Despite these limitations, it is our hope that this approach will open the door for the practical application of many existing and future image-based computer graphics techniques to video data.

Acknowledgments

We would like to thank the following people; Anat Levin, Henning Zimmer, Kaiming He, and Jiawen Chen for enabling us to compute related work comparisons by sharing their source code or binaries, Asmaa Hosni and Sebastian Volz for running their methods on our datasets, and Henning Zimmer, Peter Kaufmann, and Niko Stefanoski for helpful conversations and advice.

References

BAKER, S., SCHARSTEIN, D., LEWIS, J. P., ROTH, S., BLACK, M. J., AND SZELISKI, R. 2011. A database and evaluation methodology for optical flow. *International Journal of Computer Vision 92*, 1, 1–31.

BHAT, P., ZITNICK, C. L., COHEN, M. F., AND CURLESS, B. 2010. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Trans. Graph. 29*, 2.

CHEN, J., PARIS, S., AND DURAND, F. 2007. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph. 26*, 3, 103.

CRIMINISI, A., SHARP, T., ROTHER, C., AND PÉREZ, P. 2010. Geodesic image and video editing. *ACM Trans. Graph. 29*, 5, 134.

DOLSON, J., BAEK, J., PLAGEMANN, C., AND THRUN, S. 2010. Upsampling range data in dynamic environments. In *CVPR*, 1141–1148.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph. 21*, 3, 257–266.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4, 69.

GUO, C., MA, Q., AND ZHANG, L. 2008. Spatio-temporal saliency detection using phase spectrum of quaternion fourier transform. In *CVPR*, IEEE Computer Society.

HE, K., SUN, J., AND TANG, X. 2010. Guided image filtering. In *ECCV (1)*, Springer, vol. 6311 of *Lecture Notes in Computer Science*, 1–14.

HÖFFKEN, M., OBERHOFF, D., AND KOLESNIK, M. 2011. Temporal prediction and spatial regularization in differential optical flow. In *ACIVS*, Springer, vol. 6915 of *Lecture Notes in Computer Science*, 576–585.

HORN, B. K. P., AND SCHUNCK, B. G. 1981. Determining optical flow. *Artif. Intell. 17*, 1-3, 185–203.

HOSNI, A., RHEMANN, C., BLEYER, M., AND GELAUTZ, M. 2011. Temporally consistent disparity and optical flow via efficient spatio-temporal filtering. In *PSIVT (1)*, Springer, vol. 7087 of *Lecture Notes in Computer Science*, 165–177.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Trans. Graph. 26*, 3, 96.

KRÄHENBÜHL, P., LANG, M., HORNUNG, A., AND GROSS, M. H. 2009. A system for retargeting of streaming video. *ACM Trans. Graph. 28*, 5.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Trans. Graph. 23*, 3, 689–694.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2006. A closed form solution to natural image matting. In *CVPR (1)*, IEEE Computer Society, 61–68.

NEHAB, D., MAXIMO, A., LIMA, R. S., AND HOPPE, H. 2011. Gpu-efficient recursive filtering and summed-area tables. *ACM Trans. Graph. 30*, 6, 176.

PARIS, S., KORNPROBST, P., AND TUMBLIN, J. 2009. *Bilateral Filtering*. Now Publishers Inc., Hanover, MA, USA.

PERONA, P., AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 7, 629–639.

RHEMANN, C., HOSNI, A., BLEYER, M., ROTHER, C., AND GELAUTZ, M. 2011. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, IEEE, 3017–3024.

SCHARSTEIN, D., AND SZELISKI, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision 47*, 1-3, 7–42.

SUN, D., ROTH, S., AND BLACK, M. J. 2010. Secrets of optical flow estimation and their principles. In *CVPR*, 2432–2439.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *ICCV*, 839–846.

VOLZ, S., BRUHN, A., VALGAERTS, L., AND ZIMMER, H. 2011. Modeling temporal coherence for optical flow. In *Proc. 13th International Conference on Computer Vision (ICCV)*, IEEE Computer Society Press, Barcelona.

WANG, O., LANG, M., FREI, M., HORNUNG, A., SMOLIC, A., AND GROSS, M. H. 2011. Stereobrush: Interactive 2d to 3d conversion using discontinuous warps. In *SBM*, Eurographics Association, 47–54.

WILDEBOER, M. O., YENDO, T., TEHRANI, M. P., AND TANIMOTO, M. 2010. A semi-automatic multi-view depth estimation method. *Proceedings of SPIE, Visual Communications and Image Processing 7744*.

XIAO, J., CHENG, H., SAWHNEY, H. S., RAO, C., AND ISNARDI, M. A. 2006. Bilateral filtering-based optical flow estimation with occlusion detection. In *ECCV (1)*, Springer, vol. 3951 of *Lecture Notes in Computer Science*, 211–224.

YANG, Q., YANG, R., DAVIS, J., AND NISTÉR, D. 2007. Spatial-depth super resolution for range images. In *CVPR*, IEEE Computer Society.

ZIMMER, H., BRUHN, A., AND WEICKERT, J. 2011. Optic flow in harmony. *International Journal of Computer Vision 93*, 3, 368–388.

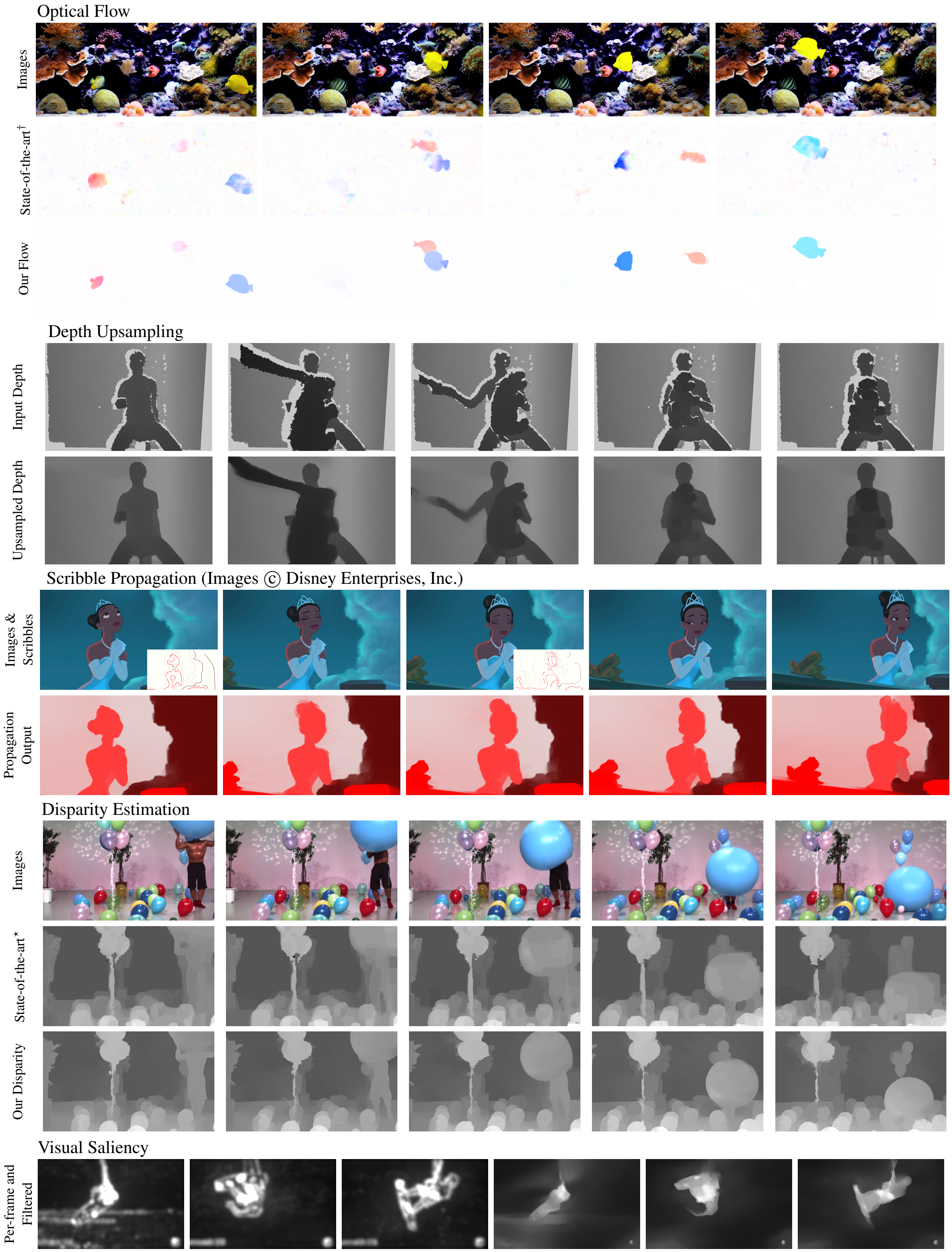

Figure 8: *Here we present a table of our results. Optical flow is compared to a high ranking (10th in the Middlebury evaluation at time of submission) per-frame method (†) [Zimmer et al. 2011]. Sparse depth data from a Microsoft Kinect is filtered to remove temporal noise and fill unknown regions. Scribble propagation is computed over 35 frames of video, with the two input scribbles used shown inset. Our two-view disparity estimation is compared to official reference depth maps provided by the MPEG group for testing (*) [Wildeboer et al. 2010], computed using three input camera views. Finally, we show three images from a per-frame visual saliency method, and then our temporally stable results. Please see the included video for a better visualization of these applications.*