DOI: 10.1111/j.1467-8659.2012.03013.x
EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (2012), Number 2

SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm

Michael Tao[1], Jiamin Bai[1], Pushmeet Kohli[2], and Sylvain Paris[3]

[1]Berkeley [2]Microsoft [3]Adobe

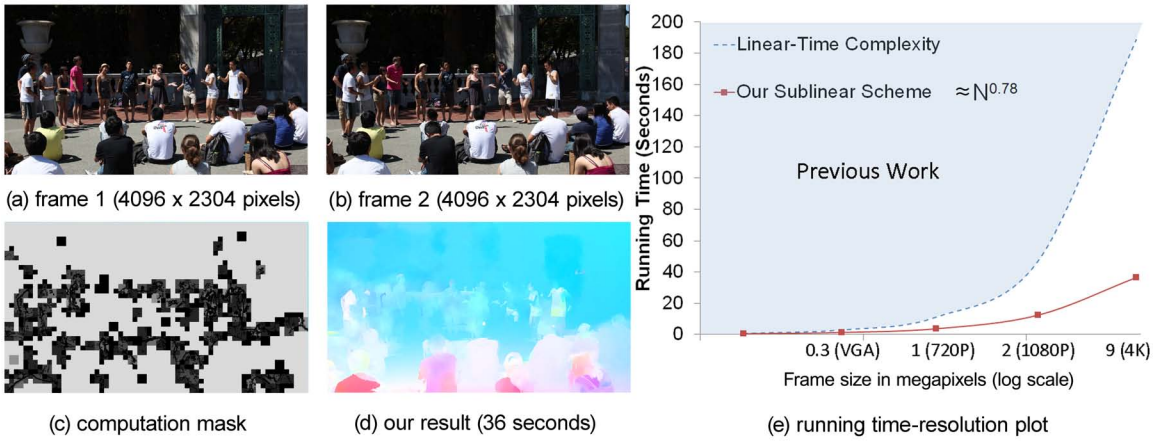


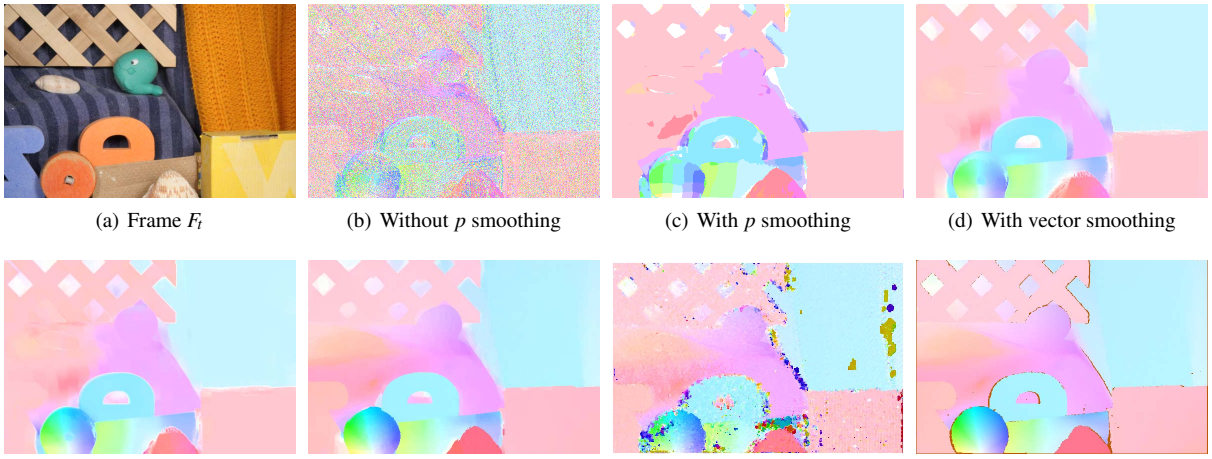
Figure 1: *The figure shows a pair of 4K video frames (a,b) and the corresponding optical flow result (d). Our new* SimpleFlow *algorithm computes the optical flow using only local operations that can be efficiently implemented on parallel architectures such as GPUs. Further, it concentrates computation where motion actually occurs, in black in (c), and uses linearly interpolation to estimate the flow in other regions, in gray in (c). This enables the computation of accurate optical maps in a reasonable amount of time (d). We show that this strategy makes the running time grow sublinearly with the frame resolution (e). This enables the processing of high-definition videos up to the 4K movie resolution in which each frame has 9 megapixels.*

Abstract

Optical flow is a critical component of video editing applications, e.g. for tasks such as object tracking, segmentation, and selection. In this paper, we propose an optical flow algorithm called SimpleFlow *whose running times increase sublinearly in the number of pixels. Central to our approach is a probabilistic representation of the motion flow that is computed using only local evidence and without resorting to global optimization. To estimate the flow in image regions where the motion is smooth, we use a sparse set of samples only, thereby avoiding the expensive computation inherent in traditional dense algorithms. We show that our results can be used as is for a variety of video editing tasks. For applications where accuracy is paramount, we use our result to bootstrap a global optimization. This significantly reduces the running times of such methods without sacrificing accuracy. We also demonstrate that the SimpleFlow algorithm can process HD and 4K footage in reasonable times.*

1. Introduction

Optical flow describes the motion in a video, i.e. it describes how each point in the scene moves from a frame to the next [BSL*07]. It is an essential component for video applications, e.g. [GGC*08, KBVF10]. Several aspects make this problem particularly challenging. (1) Occlusions: points



(a) Frame F_t (b) Without p smoothing (c) With p smoothing (d) With vector smoothing

(e) Our result alone (f) Sun et al. bootstrapped with our result (g) Lucas and Kanade (h) Ground truth

Figure 2: *With initiate frame, F_t (a), without smoothing, vectors flows are noisy (b). Using a simple bilateral filter on p produces smoother flows (c). By applying a weighted filter on the flow vectors, we achieve a subpixel flow (d). Using a coarse-to-fine approach and subpixel refinement yields our results (e). When accuracy is paramount, one can further improve the result by using our result to initialize a more computationally intensive method (f). Although our method is related in spirit to the seminal work by Lucas and Kanade [LK81] (g), it produces significantly better results (e) that are closer to the ground truth flow (h).*

can appear or disappear between two frames. (2) Aperture problem: in regions of uniform appearance, local cues do not provide any information about the motion, and only partial flow information can be recovered along one-dimensional structures.

Traditional methods for optical flow operate by matching pixels from one frame to the next based on their color. However, since there are usually many pixels of the same color in a frame, this is not a discriminative feature and thus leads to erroneous results. To increase the discriminative power, one can match blocks or group of pixels [LK81,HS81]. However, determining the size of the blocks is challenging because not all points within a block have the same displacement vector from one frame to the next.

The above-mentioned challenges associated with computing the optical flow are traditionally addressed by propagating local evidences for particular flow values across the image so that ambiguous regions get resolved due to nearby corners and textured areas. This strategy is generally implemented by defining a Markov Random Field (MRF) over the entire image. While the MRF formulation is principled and produces accurate results, finding the Maximum a Posterior (MAP) solution of the MRF is an extremely challenging problem which is computationally expensive to solve [GHN*10, RB05, LRR08].

Computing the MAP solution of the MRF involves minimizing an energy function defined over the flow variables which is an NP-hard problem. Many algorithms such as Graph Cuts [BVZ01], Loopy Belief Propagation [Pea86], and Tree Re-weighted message passing [Kol06, WJW05] have been proposed in the literature for computing good approximate solutions. The runtime complexity of these algorithms is super-linear in the number of pixels as well as in the number of labels each pixel can take, which makes them computationally prohibitive in practice.

Recently, efficient methods have been proposed such as [WPB10, BWKS06, GZG*10, SBK10]. These techniques rely on local operations performed on the GPU. To propagate information across the image domain, these schemes are iterative. While this may not be an issue with low-resolution footage, it can be a serious handicap when processing high-definition videos because many iterations are required to reach distant points. Although multiscale schemes can be used to mitigate this aspect, e.g. [BAKJH92, MP98], with high-resolution datasets such as 1080p footage (2 megapixels per frame) or 4K movies (9 megapixels), processing every pixel becomes a costly operation. Said more formally, the running times of these approaches grow linearly or superlinearly with resolution, and even optimized implementations eventually become slow because of the sheer number of pixels to process.

Rhemann et al. [RHB*11] have recently shown how edge-preserving filtering can be used for disparity estimation. This method can be extended to optical flow and is related our approach. However, there are also some key differences. This method [RHB*11] is slow, requiring more than a minute per frame at DVD resolution ($640 \times 480 \approx 0.3$ megapixel) and,

as we shall see, its running times grow superlinearly with resolution, making it ill-suited for HD footage with resolution possibly up to 9 megapixels per frame (Fig. 1).

We address these challenges with our *SimpleFlow* algorithm that is iteration-free and computes only a sparse set of samples in regions with a uniform motion. It runs in a classical multiscale fashion and at each scale, pixels are processed independently and only once. This makes our scheme highly data-parallel. This strategy ensures a linear complexity with respect to the number of pixels. Furthermore, when we upsample the data in the multiscale pyramid, we can afford to interpolate some of the flow vectors from their neighbors instead of analyzing the frame data. While strictly speaking our algorithm's complexity remains linear because we still assign a flow vector to each pixel, in practice, running times grow sublinearly because interpolation is virtually free.

The design of our method is based on a probabilistic representation of the motion at each pixel. We express the classical color invariance assumption with probabilities derived from pixel-wise color differences. The key aspect of our approach is that we manipulate these probabilities instead of flow vectors [RW97]. With this representation, the smoothness of the flow field is represented by locally averaging probabilities and accounting for edges is achieved using an edge-aware scheme such as the bilateral filter [TM98, PKTD09] that can be efficiently computed [PD09, Por08, YTA09]. Since our approach is based on weighted local color differences, it is related to the seminal work by Lucas and Kanade [LK81] and to the techniques that rely on pixel weighting, e.g. [XCS*06, YK06, Ren97]. However, unlike our method, these techniques perform costly per-pixel computation, which hampers their scalability.

While our algorithm is sufficient to achieve a reasonably accurate estimate of the motion usable for simple tasks, we can also use it to bootstrap a standard optimization-based method. This achieves state-of-the-art accuracy similar to the original optimization technique at a fraction of the cost. We demonstrate that our approach is suitable for video editing tasks such as recoloring and tracking.

Notation We consider two successive frames F_t and F_{t+1}. We use (x,y) for pixel positions and (u,v) for flow vectors, that is, we seek to estimate u and v at each pixel such that the scene point at (x,y) in F_t is visible at $(x+u,y+v)$ in F_{t+1}. Although strictly speaking, u and v depend on (x,y); for the sake of clarity, we will use the notation (u,v) instead of $\big(u(x,y),v(x,y)\big)$ when possible. We use $F_t(x,y)$ to denote the RGB color of the (x,y) pixel in F_t.

2. A Single-scale Algorithm

For sake of clarity, we first describe a version of our algorithm that operates at a single scale.

A simple likelihood model We follow the classical constant-color assumption, that is, we seek flow vectors (u,v) such that $F_t(x,y)$ and $F_{t+1}(x+u,y+v)$ are similar. We model this requirement by a simple energy term:

$$e(x,y,u,v) = \|F_t(x,y) - F_{t+1}(x+u,y+v)\|^2 \qquad (1)$$

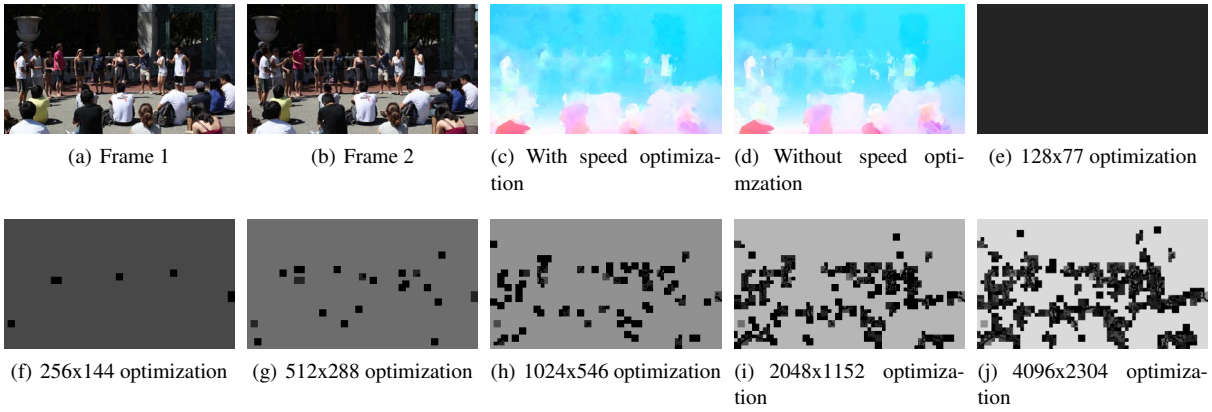
that corresponds to the likelihood probability $p \propto \exp(-e)$ [RW97]. This term is simple to compute, it is the square difference between two RGB vectors. Further, it captures nontrivial information about the local structure of the video such as the fact that edges suffer from a one-dimensional ambiguity. p also naturally represents the reliability of the information that it embeds; for instance, if p is nearly uniform, it provides only a weak evidence about the flow whereas peaked distributions indicate reliable data. Because it is a point-wise computation, p often provides noisy and ambiguous information but combining cues across pixels yields a rich representation of the flow.

Local validity as smoothness prior We also assume that the flow is locally smooth. However, we do not rely on a pairwise term such as $\|(u_1,v_1)-(u_2,v_2)\|^2$ as often used in the literature [BSL*07]. The drawback with such pairwise terms is that they link the flow estimates at several pixels, generating dependencies between unknowns that make the optimization problem harder to solve. Instead, we require that a flow vector (u_0,v_0) at a pixel (x_0,y_0) is a good explanation of the motion at (x_0,y_0) as well as other pixels in a neighborhood $\mathcal{N}_0$. A simple way to express this assumption is $(u_0,v_0) = \arg\max_{(u,v)\in\Omega} \prod_{(x,y)\in\mathcal{N}_0} p(x,y,u,v)$ where Ω is the set of possible (u,v) vectors that we consider. By pooling information from all the pixels in $\mathcal{N}_0$, this yields a smoother and cleaner estimate of the flow (Fig. 2 (c)). We can better understand this behavior in the negative log-likelihood (or energy cost) domain where we try to find the lowest cost flow vector for the pixel (x_0,y_0): $(u_0,v_0) = \arg\min_{(u,v)\in\Omega} \sum_{(x,y)\in\mathcal{N}_0} e(x,y,u,v)$. Our smoothness prior amounts to smoothing the likelihood term instead of adding a pairwise term. As a consequence, finding a minimizer remains simple since there is no interaction between the unknowns, the solution at a pixel does not depend on the solution at its neighbors.

Smoothing with a box filter would not differentiate between pixels within $\mathcal{N}_0$, e.g. it would ignore edges. We add weights to account for how pixels relate to each other. In this paper, we apply two weighting functions, w_d for the distance between pixels and w_c for the color difference. We propose the following log-likelihood:

$$E(x_0,y_0,u,v) = \sum_{(x,y)\in\mathcal{N}_0} w_\mathrm{d}\, w_\mathrm{c}\, e(x,y,u,v) \qquad (2)$$
$$\text{with } w_\mathrm{d} = \exp\left(-\|(x_0,y_0)-(x,y)\|^2/2\sigma_\mathrm{d}\right)$$
$$\text{and } w_\mathrm{c} = \exp\left(-\|F_t(x_0,y_0)-F_t(x,y)\|^2/2\sigma_\mathrm{c}\right)$$

As shown in Figure 2 (d), adding weights improves the results. Further, the weights correspond to the bilateral filter



(a) Frame 1 (b) Frame 2 (c) With speed optimization (d) Without speed optimzation (e) 128x77 optimization

(f) 256x144 optimization (g) 512x288 optimization (h) 1024x546 optimization (i) 2048x1152 optimization (j) 4096x2304 optimization

Figure 3: *Our sublinear scheme runs a full flow estimation only at a few key pixels at each scale. In this 4K HD example (*4096×2304*), we use six scales (e,f,g,h,i,j,k). Black indicates where we run a full estimation. Brighter shades correspond to regions where we estimate the flow with linear interpolation on* 2×2*,* 4×4*,* 16×16*,* 32×32*, and* 64×64 *windows respectively. As illustrated on this example, our scheme concentrates most of the computation in discontinuous areas, thereby drastically reducing the required computational effort (c) and produces similar results compared to using no optimization (d) (Fig. 4).*

weights [PKTD09], which makes it possible to compute E with optimized schemes, e.g. [PD09, CPD07, AGDL09, ABD10, Por08, YTA09].

Detecting occlusions To detect occlusions, we compare the forward flow (u_f, v_f) from t to $t+1$, and the backward flow (u_b, v_b) from $t+1$ to t. Ideally, one should be the opposite of the other. One can either test the equality to get a binary detector or compute the difference $\|(u_f, v_f) - (-u_b, -v_b)\|$ for a continuous estimator. When the computed difference is high, we mark the pixel as occluded between the frames.

Practical implementation For each pixel (x_0, y_0), our prototype implementation computes e on $n \times n$ windows centered on (x_0, y_0), i.e., we compute the color difference between $F_t(x_0, y_0)$ and every pixel in a $n \times n$ window in F_{t+1}. This produces a n^2-dimensional vector $\mathbf{e}$ at each pixel. Then, we compute E by applying a bilateral filter on these $\mathbf{e}$ vectors using the frame data F_t to define the color weights w_c. This is known as a cross- or joint-bilateral filtering [ED04, PAH*04] and can be sped up with optimized data structures, e.g. [PD09, CPD07, AGDL09, ABD10]. Finally, we estimate the flow as the (u_0, v_0) vector that minimizes E. This produces values that are integers; to get a subpixel estimate, we fit a parabola to the 3×3 pixels centered at (u_0, v_0) and extract its minimum. We found it useful to further regularize the result by applying a bilateral filter on the flow vectors. For this operation, we discard the occluded pixels, and use the weights w_d and w_c with an additional weight w_r that represents how reliable is our flow estimate at (x, y):

$$w_r(x,y) = \underset{(u,v)\in\Omega}{\text{mean}}\, e(x,y,u,v) - \underset{(u,v)\in\Omega}{\min}\, e(x,y,u,v)$$

Discussion This scheme is not efficient because we need to consider large $n \times n$ windows to be able to recover large motions. We address this issue in the next section with a multiscale approach. Representing the motion by the likelihood p is related to the work of Rosenberg and Werman [RW97]. While they already underscore the usefulness and richness of this probability distribution, they use it for tracking discrete points. In comparison, we focus on dense flow field and use these probabilities in conjunction with edge-aware filtering and multiscale processing. Although we use the bilateral filter to aggregate local evidences about the flow, we believe that other edge-aware schemes would perform equivalently. We see the exploration of other options as a possible direction for future work.

3. A Multiscale Sublinear Algorithm

This section describes how we make our simple algorithm faster using an adaptive multi-scale strategy. First, we show how to structure the algorithm described in the previous section in a multi-scale fashion. We then describe how to make it efficient to that its running times grow sublinearly with respect to the frame resolution of the video.

3.1. Multiscale Flow Estimation

We construct an image pyramid for each image frame in which each level is twice coarser than the previous one, i.e. the ℓ-th level has a resolution 2^ℓ times lower than the original frame. At the coarsest level of the pyramid, we estimate the flow using the scheme described in Section 2. We now explain how to compute the flow at level ℓ assuming the flow at $\ell + 1$ is known.

We generate an initial estimate $(\bar{u}_\ell, \bar{v}_\ell)$ of the flow by upsampling the flow from the previous level $(u_{\ell+1}, v_{\ell+1})$ using joint-bilateral upsampling [KCLU07], followed by a multiplication by 2 to account for the change in resolution. We then follow our standard flow computation process described in Section 2 with the only change that now we aggregate probabilities in a neighborhood $\bar{\mathcal{N}}$ centered on $(x_0+\bar{u}, y_0+\bar{v})$. For the flow vector at (x_0, y_0), we select (u_0, v_0) that minimizes the log-likelihood $E(x_0, y_0, u, v)$.

3.2. An Efficient Scheme

The pyramid scheme described in the previous section is a standard multiscale approach, and has a linear complexity with respect to the number of pixels. We now describe our adaptive multi-scale strategy which tries to isolate image regions where the flow changes slowly. For pixels in such regions, we replace the computationally expensive energy minimization operation by a simple interpolation operation. Although this is not strictly speaking a sublinear scheme since we still process each pixel, it behaves so in practice because interpolation has a negligible cost.

For each layer, we estimate a flow irregularity map where the flow is smooth and where it varies more. At each pixel (x_0, y_0), we compute the irregularity value as $\max_{(x,y)\in\mathcal{N}_0} \| \big(u(x,y), v(x,y)\big) - \big(u(x_0,y_0), v(x_0,y_0)\big) \|$. During upscaling, if this value is above a threshold τ, we run the full pipeline on the corresponding upscaled pixels. Otherwise, we compute the flow at the corners of the patch, and find the flows at other pixel using bilinear interpolation. We recursively find smooth flow regions as we upscale, which effectively yields constant time computation to upsample these areas. We found that this bilinear scheme is fast and works well. But when more accuracy is desirable, one could trade-off estimating the flow at more points and fitting a higher-order function to them. Evaluating this option is kept as future work.

4. Results

We conducted several experiments to evaluate the performance of our method and its components. The data used for the experiments comprises of a number of image pairs, including some from the Middlebury optical flow benchmark. The values used for the different parameters of the algorithm were: $\sigma_c = 0.08$, $\sigma_d = 5.5$, $\tau = 0.25$, $\mathcal{N}$ a 11×11 window and Ω a 20×20 window.

Accuracy To evaluate our accuracy, we submitted to the Middlebury optical flow benchmark proposed by Baker et al. [BSL*07]. Figure 6 shows that our algorithm performs as one of the most accurate algorithms, ranking 10th overall for average end point error and 6th for angle error. We used the method of Sun et al. [SRB10] to refine our results. For example, on Urban3 of the Middlebury dataset, our algorithm computes the initialization, saving computational time by a factor of around 43× (1.7 seconds versus 73.8 seconds, using our hardware). Because the convergence is nearly the same across all examples and the penalty in accuracy is marginal (Fig. 6 and 8), our algorithm provides a significant benefit in computational efficiency, allowing the main part of the algorithm to have a sublinear computational cost in practice. Iterations of the refining algorithms after the first initial guess require the same time as a normal run. We only use this refinement strategy for our Middlebury examples. As we show in the next section, we found that the results produced by our algorithm alone are sufficient in practice for a number of applications.

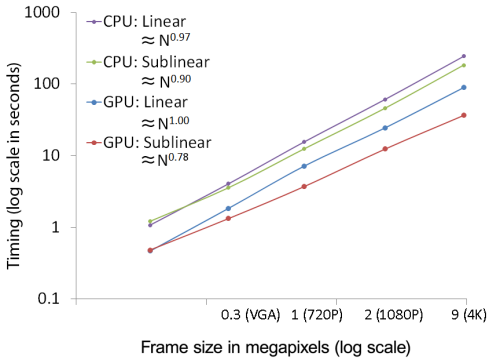


Figure 4: *Log-log plot of the running times with respect to the video size. Here shows how our algorithm exploits parallelized architecture such as the GPU. Our sublinear method performs better on the GPU and most importantly yields running times that grow as $N^{0.78}$ (in blue).*

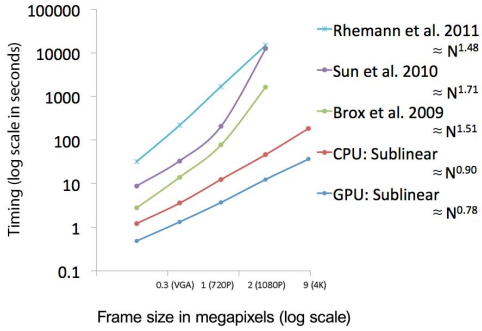


Figure 5: *Log-log plot of the running times with respect to the video size. Comparative to other algorithms, our running times grow sublinearly relative to number pixels while others compute at nonlinear rates. We recorded timings for Rhemann et al. [RHB*11], Sun et al. [SRB10], and Brox et al. [BBM09] because they took too long to complete at 4K resolutions.*

Running Time We also verified that our complete scheme actually behaves sublinearly. Figure 4 shows a plot of its

Average angle error	avg. rank	Army (Hidden texture) GT all	Army (Hidden texture) GT disc	Army (Hidden texture) GT untext	Army (Hidden texture) im0 all	Army (Hidden texture) im0 disc	Army (Hidden texture) im0 untext	Army (Hidden texture) im1 all	Army (Hidden texture) im1 disc	Army (Hidden texture) im1 untext
Layers++ [38]	6.1	3.11 2	8.22 2	2.79 6	2.43 5	**7.02** 1	2.24 11	**2.43** 1	**5.77** 1	2.18 11
MDP-Flow2 [40]	6.1	3.32 7	8.76 5	2.85 8	**2.18** 1	7.47 3	1.85 4	2.77 3	6.95 3	2.06 9
LSM [41]	7.5	3.12 3	8.62 3	2.75 4	3.00 14	10.5 12	2.44 16	3.43 7	8.85 8	2.35 13
TC-Flow [48]	7.8	**2.91** 1	**8.00** 1	2.34 2	**2.18** 1	8.77 7	**1.52** 1	3.84 12	10.7 13	**1.49** 1
Classic+NL [31]	8.8	3.20 5	8.72 4	2.81 7	3.02 15	10.6 13	2.44 16	3.46 9	8.84 7	2.38 15
SimpleFlow [52]	10.3	3.35 8	9.20 8	2.98 10	3.18 17	10.7 15	2.71 22	5.06 18	12.6 17	2.70 18
OF-MoI [49]	11.8	3.19 4	8.76 5	2.77 5	3.84 26	14.0 29	2.69 21	3.44 8	8.78 6	2.39 16
MDP-Flow [26]	12.4	3.48 11	9.46 12	3.10 14	2.45 6	7.36 2	2.41 14	3.21 5	8.31 5	2.78 20
CostFilter [42]	12.6	3.84 13	9.64 14	3.06 12	2.55 9	8.09 5	2.03 7	2.69 2	6.47 2	1.88 4
OFH [39]	13.9	3.90 16	9.77 16	3.62 24	2.84 12	11.0 17	2.04 8	5.52 21	14.4 22	1.89 5

Average angle error	avg. rank	Mequon (Hidden texture) GT all	Mequon (Hidden texture) GT disc	Mequon (Hidden texture) GT untext	Mequon (Hidden texture) im0 all	Mequon (Hidden texture) im0 disc	Mequon (Hidden texture) im0 untext	Mequon (Hidden texture) im1 all	Mequon (Hidden texture) im1 disc	Mequon (Hidden texture) im1 untext
Layers++ [38]	6.1	**2.13** 1	**9.71** 1	**1.15** 1	**2.35** 1	**3.02** 1	**1.96** 1	3.81 10	11.4 7	3.22 13
MDP-Flow2 [40]	6.1	3.25 13	17.3 15	1.59 11	2.87 6	3.73 5	2.32 5	3.15 3	11.1 6	2.65 3
LSM [41]	7.5	2.66 3	13.6 3	1.44 2	2.82 4	3.68 3	2.36 7	3.38 5	9.41 3	2.81 8
TC-Flow [48]	7.8	3.13 10	16.6 12	1.46 4	2.78 3	3.73 5	**1.96** 1	3.08 2	11.4 7	2.66 4
Classic+NL [31]	8.8	2.78 4	14.3 4	1.46 4	2.83 5	3.68 3	2.31 4	3.40 6	**9.09** 1	2.76 7
SimpleFlow [52]	10.3	2.95 6	15.1 7	1.58 9	2.91 7	3.79 7	2.47 10	3.59 9	9.49 4	2.99 10
OF-MoI [49]	11.8	2.98 7	15.8 9	1.53 7	2.96 8	3.89 9	2.34 6	3.40 6	9.30 2	2.73 5
MDP-Flow [26]	12.4	3.18 12	17.8 17	1.70 13	3.03 9	3.87 8	2.60 14	3.43 8	12.6 11	2.81 8
CostFilter [42]	12.6	3.66 18	16.8 13	1.88 17	2.62 2	3.34 2	1.99 3	4.05 15	11.0 5	3.65 20
OFH [39]	13.9	3.52 14	20.5 26	1.60 12	3.18 11	4.06 11	2.82 19	3.86 11	14.1 18	3.59 17

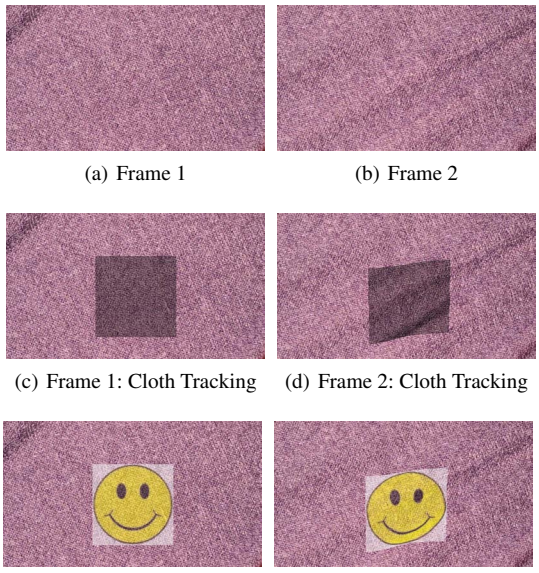
Average angle error	avg. rank	Schefflera (Hidden texture) GT all	Schefflera (Hidden texture) GT disc	Schefflera (Hidden texture) GT untext	Schefflera (Hidden texture) im0 all	Schefflera (Hidden texture) im0 disc	Schefflera (Hidden texture) im0 untext	Schefflera (Hidden texture) im1 all	Schefflera (Hidden texture) im1 disc	Schefflera (Hidden texture) im1 untext
Layers++ [38]	6.1	**2.43** 1	**5.77** 1	2.18 11	**2.13** 1	**9.71** 1	**1.15** 1	**2.35** 1	**3.02** 1	**1.96** 1
MDP-Flow2 [40]	6.1	2.77 3	6.95 3	2.06 9	3.25 13	17.3 15	1.59 11	2.87 6	3.73 5	2.32 5
LSM [41]	7.5	3.43 7	8.85 8	2.35 13	2.66 3	13.6 3	1.44 2	2.82 4	3.68 3	2.36 7
TC-Flow [48]	7.8	3.84 12	10.7 13	**1.49** 1	3.13 10	16.6 12	1.46 4	2.78 3	3.73 5	**1.96** 1
Classic+NL [31]	8.8	3.46 9	8.84 7	2.38 15	2.78 4	14.3 4	1.46 4	2.83 5	3.68 3	2.31 4
SimpleFlow [52]	10.3	5.06 18	12.6 17	2.70 18	2.95 6	15.1 7	1.58 9	2.91 7	3.79 7	2.47 10
OF-MoI [49]	11.8	3.44 8	8.78 6	2.39 16	2.98 7	15.8 9	1.53 7	2.96 8	3.89 9	2.34 6
MDP-Flow [26]	12.4	3.21 5	8.31 5	2.78 20	3.18 12	17.8 17	1.70 13	3.03 9	3.87 8	2.60 14
CostFilter [42]	12.6	2.69 2	6.47 2	1.88 4	3.66 18	16.8 13	1.88 17	2.62 2	3.34 2	1.99 3
OFH [39]	13.9	5.52 21	14.4 22	1.89 5	3.52 14	20.5 26	1.60 12	3.18 11	4.06 11	2.82 19

Average angle error	avg. rank	Wooden (Hidden texture) GT all	Wooden (Hidden texture) GT disc	Wooden (Hidden texture) GT untext	Wooden (Hidden texture) im0 all	Wooden (Hidden texture) im0 disc	Wooden (Hidden texture) im0 untext	Wooden (Hidden texture) im1 all	Wooden (Hidden texture) im1 disc	Wooden (Hidden texture) im1 untext
Layers++ [38]	6.1	**2.13** 1	**9.71** 1	**1.15** 1	**2.35** 1	**3.02** 1	**1.96** 1	3.81 10	11.4 7	3.22 13
MDP-Flow2 [40]	6.1	3.25 13	17.3 15	1.59 11	2.87 6	3.73 5	2.32 5	3.15 3	11.1 6	2.65 3
LSM [41]	7.5	2.66 3	13.6 3	1.44 2	2.82 4	3.68 3	2.36 7	3.38 5	9.41 3	2.81 8
TC-Flow [48]	7.8	3.13 10	16.6 12	1.46 4	2.78 3	3.73 5	**1.96** 1	3.08 2	11.4 7	2.66 4
Classic+NL [31]	8.8	2.78 4	14.3 4	1.46 4	2.83 5	3.68 3	2.31 4	3.40 6	**9.09** 1	2.76 7
SimpleFlow [52]	10.3	2.95 6	15.1 7	1.58 9	2.91 7	3.79 7	2.47 10	3.59 9	9.49 4	2.99 10
OF-MoI [49]	11.8	2.98 7	15.8 9	1.53 7	2.96 8	3.89 9	2.34 6	3.40 6	9.30 2	2.73 5
MDP-Flow [26]	12.4	3.18 12	17.8 17	1.70 13	3.03 9	3.87 8	2.60 14	3.43 8	12.6 11	2.81 8
CostFilter [42]	12.6	3.66 18	16.8 13	1.88 17	2.62 2	3.34 2	1.99 3	4.05 15	11.0 5	3.65 20
OFH [39]	13.9	3.52 14	20.5 26	1.60 12	3.18 11	4.06 11	2.82 19	3.86 11	14.1 18	3.59 17

Average angle error	avg. rank	Grove (Synthetic) GT all	Grove (Synthetic) GT disc	Grove (Synthetic) GT untext	Grove (Synthetic) im0 all	Grove (Synthetic) im0 disc	Grove (Synthetic) im0 untext	Grove (Synthetic) im1 all	Grove (Synthetic) im1 disc	Grove (Synthetic) im1 untext
Layers++ [38]	6.1	**2.35** 1	**3.02** 1	**1.96** 1	3.81 10	11.4 7	3.22 13	**2.43** 1	**5.77** 1	2.18 11
MDP-Flow2 [40]	6.1	2.87 6	3.73 5	2.32 5	3.15 3	11.1 6	2.65 3	2.77 3	6.95 3	2.06 9
LSM [41]	7.5	2.82 4	3.68 3	2.36 7	3.38 5	9.41 3	2.81 8	3.43 7	8.85 8	2.35 13
TC-Flow [48]	7.8	2.78 3	3.73 5	**1.96** 1	3.08 2	11.4 7	2.66 4	3.84 12	10.7 13	**1.49** 1
Classic+NL [31]	8.8	2.83 5	3.68 3	2.31 4	3.40 6	**9.09** 1	2.76 7	3.46 9	8.84 7	2.38 15
SimpleFlow [52]	10.3	2.91 7	3.79 7	2.47 10	3.59 9	9.49 4	2.99 10	5.06 18	12.6 17	2.70 18
OF-MoI [49]	11.8	2.96 8	3.89 9	2.34 6	3.40 6	9.30 2	2.73 5	3.44 8	8.78 6	2.39 16
MDP-Flow [26]	12.4	3.03 9	3.87 8	2.60 14	3.43 8	12.6 11	2.81 8	3.21 5	8.31 5	2.78 20
CostFilter [42]	12.6	2.62 2	3.34 2	1.99 3	4.05 15	11.0 5	3.65 20	2.69 2	6.47 2	1.88 4
OFH [39]	13.9	3.18 11	4.06 11	2.82 19	3.86 11	14.1 18	3.59 17	5.52 21	14.4 22	1.89 5

Average angle error	avg. rank	Urban (Synthetic) GT all	Urban (Synthetic) GT disc	Urban (Synthetic) GT untext	Urban (Synthetic) im0 all	Urban (Synthetic) im0 disc	Urban (Synthetic) im0 untext	Urban (Synthetic) im1 all	Urban (Synthetic) im1 disc	Urban (Synthetic) im1 untext
Layers++ [38]	6.1	**3.81** 10	11.4 7	3.22 13	**2.13** 1	**9.71** 1	**1.15** 1	**2.35** 1	**3.02** 1	**1.96** 1
MDP-Flow2 [40]	6.1	3.15 3	11.1 6	2.65 3	3.25 13	17.3 15	1.59 11	2.87 6	3.73 5	2.32 5
LSM [41]	7.5	3.38 5	9.41 3	2.81 8	2.66 3	13.6 3	1.44 2	2.82 4	3.68 3	2.36 7
TC-Flow [48]	7.8	3.08 2	11.4 7	2.66 4	3.13 10	16.6 12	1.46 4	2.78 3	3.73 5	**1.96** 1
Classic+NL [31]	8.8	3.40 6	**9.09** 1	2.76 7	2.78 4	14.3 4	1.46 4	2.83 5	3.68 3	2.31 4
SimpleFlow [52]	10.3	3.59 9	9.49 4	2.99 10	2.95 6	15.1 7	1.58 9	2.91 7	3.79 7	2.47 10
OF-MoI [49]	11.8	3.40 6	9.30 2	2.73 5	2.98 7	15.8 9	1.53 7	2.96 8	3.89 9	2.34 6
MDP-Flow [26]	12.4	3.43 8	12.6 11	2.81 8	3.18 12	17.8 17	1.70 13	3.03 9	3.87 8	2.60 14
CostFilter [42]	12.6	4.05 15	11.0 5	3.65 20	3.66 18	16.8 13	1.88 17	2.62 2	3.34 2	1.99 3
OFH [39]	13.9	3.86 11	14.1 18	3.59 17	3.52 14	20.5 26	1.60 12	3.18 11	4.06 11	2.82 19

Figure 6: *Bootstrapping the Sun et al. algorithm with our result achieves high accuracy results while reducing computation time. Our algorithm ranks as one of the highest in the Middlebury evaluation and the error difference among the top algorithms are marginal as seen in their metrics.*



(a) Frame 1 (b) Frame 2

(c) Frame 1: Cloth Tracking (d) Frame 2: Cloth Tracking

(e) Frame 1: Texture Mapping (f) Frame 2: Texture Mapping

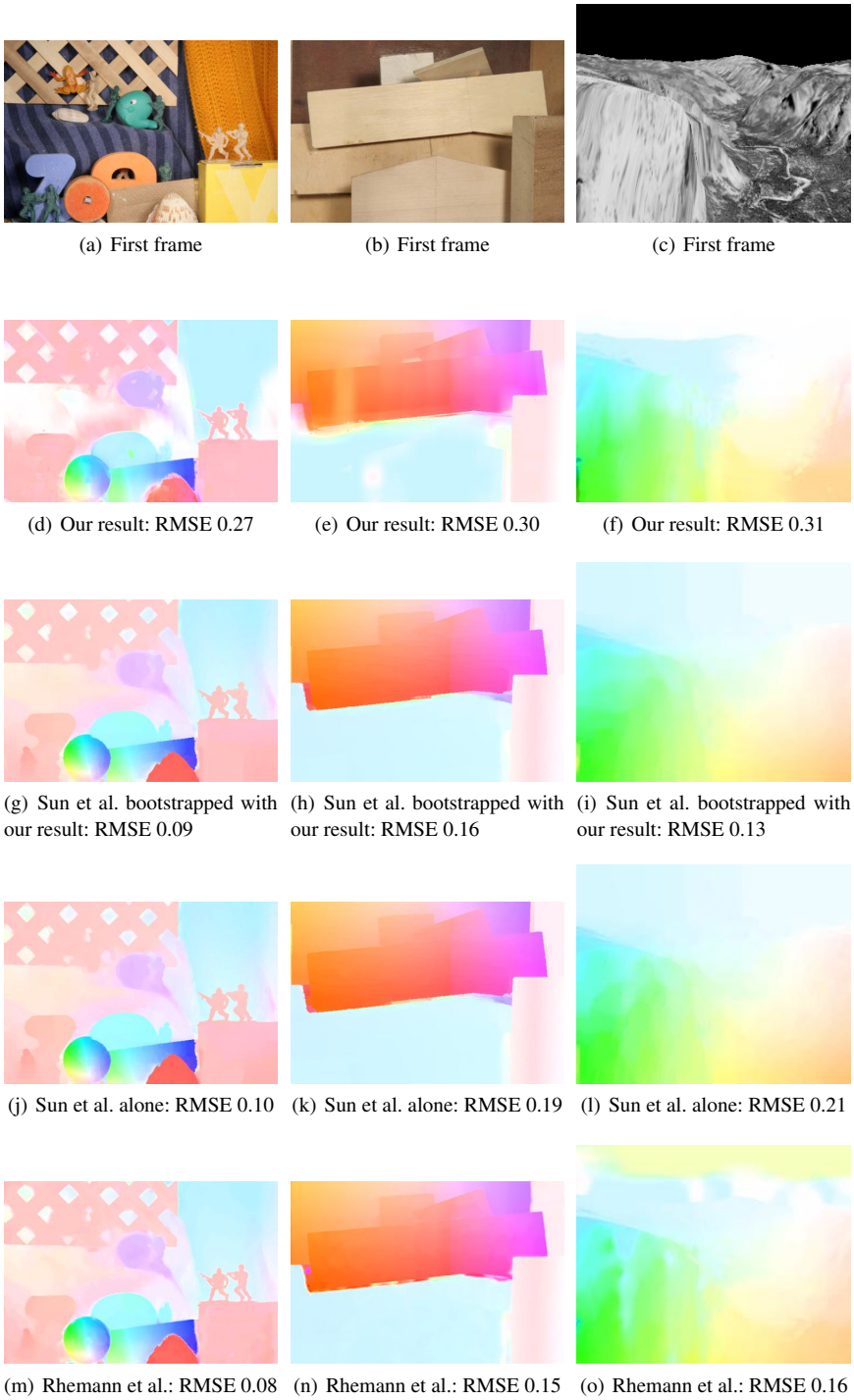
Figure 7: *The figure shows the results of cloth tracking in a full HD 720 video. We use the tracks to warp an overlaid texture.*

running time for various resolutions obtained by using the same video pair frames from 4K (4096×2304) and downscaling the frames by factors of 2. In this log-log plot, the slope of the curve indicates the complexity of the algorithm. We measure of slope of 0.78 which means that empirically, our running times grow as $N^{0.78}$ with N the number of pixels in a frame. The degree of sublinearity depends on the actual video being processed, the more it contains regions with smooth flow, the better it behaves. We give statistics about more videos in supplemental material. In the worst case with an extremely irregular flow, our algorithm would behave linearly, which is still acceptable. However, we never experienced such pathological case in practice. Typical running time for the Middlebury examples was 1.6 seconds with a resolution of 584×388. We compare our results with two recent works on optical flow proposed by Sun et al. [SRB10] and Brox et al. [BBM09] (Fig. 5). The comparison shows how our algorithm significantly reduces the computation time, making the processing of 4K footage reasonable whereas it is prohibitively expensive with other approaches. The large performance increase is due to the fact that we process our each pixel independently, which enables the effective use of parallel architectures. Full analysis on our architecture performance scaling with threads and GPU use is provided in supplementary material. The scalability of our algorithm makes it a premium choice for processing very high resolution footage as used in the movie industry.

Limitations While our evaluation shows that our algorithm performs well in general, our choice to forgo global optimization can make it difficult to handle scenes with repeated patterns. Because our method only considers local neighborhoods, it may not be able differentiate between repetitions of the pattern whereas a global technique might be able to gather evidence from a larger support to resolve the ambiguity. However, we only observed this issue on the synthetic *urban* scenes of the Middlebury benchmark. The other failure case that we observe is due to fast motion. Our method is unable to cope with objects moving fast that appear far apart in the two and which appearance changes a lot (Fig. 10). Also, our algorithm may miss small objects in the middle of regions with uniform regions in which we resort to interpolation instead of flow estimation.

4.1. Demo Applications

To demonstrate that our results can be used to edit videos, we implemented a couple of standard video editing tasks such as object recoloring and tracking using the optical flow computed using the SimpleFlow algorithm. We do not claim these applications as contributions of this work. We acknowledge that one could add more sophistication to them, for instance to deal with occlusions but this is beyond the scope of this paper. These applications are only provided as



(a) First frame (b) First frame (c) First frame

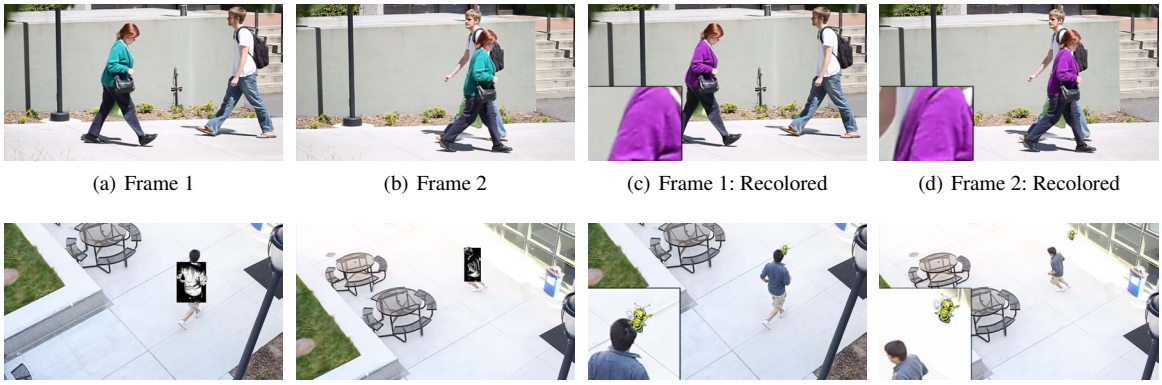
(d) Our result: RMSE 0.27 (e) Our result: RMSE 0.30 (f) Our result: RMSE 0.31

(g) Sun et al. bootstrapped with our result: RMSE 0.09 (h) Sun et al. bootstrapped with our result: RMSE 0.16 (i) Sun et al. bootstrapped with our result: RMSE 0.13

(j) Sun et al. alone: RMSE 0.10 (k) Sun et al. alone: RMSE 0.19 (l) Sun et al. alone: RMSE 0.21

(m) Rhemann et al.: RMSE 0.08 (n) Rhemann et al.: RMSE 0.15 (o) Rhemann et al.: RMSE 0.16

Figure 8: *Results on datasets from the Middlebury benchmark [BSL*07]. Our algorithm achieves results that are good enough for graphics applications as demonstrated in the companion video. If more accuracy is needed, one can use our results to bootstrap a global optimization method such as the one by Sun et al. [SRB10]. This produces better results while saving a significant amount of time compared to running a global optimization using a naive initialization. These results are similar to the output of the algorithm by Rhemann et al. [RHB*11] that also uses local operations while being a lot faster because we linearly interpolate the flow where it is smooth, whereas Rhemann's technique runs a full estimation at each pixel.*



(a) Frame 1 (b) Frame 2 (c) Frame 1: Recolored (d) Frame 2: Recolored

(e) Frame 1: Tracking (f) Frame 2: Tracking (g) Frame 1: Object Placement (h) Frame 2: Object Placement

Figure 9: *The figure shows the results of some typical editing operations performed using the optical flow obtained from SimpleFlow in full HD 720P videos. The first row is an application for recoloring. We show two different frames (a,b). We selected the shirt to be recolored and propagation occurs throughout the video. The results are seamless and produce high quality results (c,d). With a similar approach, we use our dense vector field for tracking (c,d). Editing applications such as dynamic object placement use these fields to generate objects following the tracked object (e,f).*

(a) Frame 1 (b) Frame 2 (c) Our Result (d) Lucas and Kanade

Figure 10: *Failure case. These two frames exhibit a large displacement (a,b). While our method produces acceptable results on the background and the dog body, it does not capture the fast-moving legs (c). In comparison, the Lucas-Kanade method [LK81] completely fails (d). Because of its very local nature, it is unable to handle the large motion in this sequence. Note that we used the same color code in (c) and (d).*

a demonstration of what one can build using our optical flow method. The recoloring and tracking results obtained by our algorithm are shown in Figure 9.

Recoloring We allow users to change the color of an object. First, they select a region to recolor. In this selection, we select the central part and expand it using simple color similarity. We then compute the average flow in this region and use it to advect the selection to the next frame where we repeat the same process. Selection expansion and advection are done using a bilateral filter scheme that accounts for the color and similarities.

Tracking and Dynamic Object Placement This application essentially uses the same mechanism as the recoloring. To place an object near the subject, we simply use these flow vectors to compute where the object should move about the subject.

Cloth texturing We use our optical flow to advect a grid of anchor points at the surface of deforming piece of cloth. The motion of each point is estimated by averaging the flow vectors in a small neighborhood. This produces a dense grid which we use to add a texture on top of the garment. The results obtained by our algorithm are shown in Figure 7.

5. Discussion and Conclusion

We presented a simple method for optical flow with running times that grow sublinearly with video resolution. A key property of our approach is that we do not resort to global optimization to propagate local information across the image. Instead, we average local probability distributions computed from standard color differences. The fact that we recover an accurate flow field from such simple cues suggests that they contain actually more information about the scene than what their simplicity suggests at first. The local aspect of our scheme is also the key component that enables sublinear computation. We believe that this property is critical to be able process high-resolution footage such as HD videos and movie sequences in their original format.

References

[ABD10] ADAMS A., BAEK J., DAVIS A.: Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum* (2010). Proceedings of the Eurographics conference. 3, 4

[AGDL09] ADAMS A., GELFAND N., DOLSON J., LEVOY M.: Gaussian KD-trees for fast high-dimensional filtering. *ACM*

Transactions on Graphics 28, 3 (2009). Proceedings of the ACM SIGGRAPH conference. 3, 4

[BAKJH92] BERGEN J. R., ANANDAN P., KEITH J. HANNA R. H.: Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision* (1992). 2

[BBM09] BROX T., BREGLER C., MALIK J.: Large displacement optical flow. *IEEE Conference on Computer Vision and Pattern Recongition* (2009). 5, 6

[BSL*07] BAKER S., SCHARSTEIN D., LEWIS J., ROTH S., BLACK M., SZELISKI R.: A database and evaluation methodology for optical flow. In *Proceedings of the International Conference on Computer Vision* (2007). 1, 3, 6, 7

[BVZ01] BOYKOV Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2001). 2

[BWKS06] BRUHN A., WEICKERT J., KOHLBERGER T., SCHNÖRR C.: A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. *International Journal of Computer Vision 70*, 3 (2006). 2

[CPD07] CHEN J., PARIS S., DURAND F.: Real-time edge-aware image processing with the bilateral grid. *ACM Transactions on Graphics 26*, 3 (2007). Proceedings of the ACM SIGGRAPH conference. 3, 4

[ED04] EISEMANN E., DURAND F.: Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics 23*, 3 (2004). Proceedings of the ACM SIGGRAPH conference. 4

[GGC*08] GOLDMAN D. B., GONTERMAN C., CURLESS B., SALESIN D., SEITZ S. M.: Video object annotation, navigation, and composition. In *Proceedings of ACM symposuim on User Interface Software and Technology* (2008). 1

[GHN*10] GLOCKER B., HEIBEL H., NAVAB N., KOHLI P., ROTHER C.: Triangleflow: Optical flow with triangulation-based higher-order likelihoods. In *Proceedings of the European Conference on Computer Vision* (2010). 2

[GZG*10] GWOSDEK P., ZIMMER H., GREWENIG S., BRUHN A., WEICKERT J.: A highly efficient GPU implementation for variational optic flow based on the Euler-Lagrange framework. In *Proceedings of the Workshop for Computer Vision with GPUs* (2010). 2

[HS81] HORN B. K., SCHUNCK B. G.: Determining optical flow. *Artificial Intelligence 17* (1981). 2

[KBVF10] KUETTEL D., BREITENSTEIN M. D., VAN GOOL L., FERRARI V.: What's going on? Discovering spatio-temporal dependencies in dynamic scenes. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (2010), IEEE. 1

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. *ACM Transactions on Graphics 26*, 3 (2007). Proceedings of the ACM SIGGRAPH conference. 4

[Kol06] KOLMOGOROV V.: Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 10 (2006), 1568–1583. 2

[LK81] LUCAS B. D., KANADE T.: An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop* (1981). 2, 3, 8

[LRR08] LEMPITSKY V., ROTH S., ROTHER C.: Fusionflow: discrete-continuous optimization for optical flow estimation. In *IEEE Conference on Computer Vision and Pattern Recongition* (2008). 2

[MP98] MÉNIN E., PÉREZ P.: A multigrid approach for hierarchical motion estimation. In *Proceedings of the IEEE International Conference on Computer Vision* (1998). 2

[PAH*04] PETSCHNIGG G., AGRAWALA M., HOPPE H., SZELISKI R., COHEN M., TOYAMA K.: Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics 23*, 3 (July 2004). Proceedings of the ACM SIGGRAPH conference. 4

[PD09] PARIS S., DURAND F.: A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision* (2009). 3, 4

[Pea86] PEARL J.: Fusion, propagation, and structuring in belief networks. *Artificial Intelligence 29*, 3 (1986), 241–288. 2

[PKTD09] PARIS S., KORNPROBST P., TUMBLIN J., DURAND F.: Bilateral filtering: Theory and applications. Foundations and Trends in Computer Graphics and Vision, 2009. 3

[Por08] PORIKLI F.: Constant time O(1) bilateral filtering. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (2008). 3

[RB05] ROTH S., BLACK M.: Fields of experts: a framework for learning image priors. In *IEEE Conference on Computer Vision and Pattern Recongition* (2005). 2

[Ren97] REN X.: Local grouping for optical flow. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (1997). 3

[RHB*11] RHEMANN C., HOSNI A., BLEYER M., ROTHER C., GELAUTZ M.: Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (2011). 2, 5, 7

[RW97] ROSENBERG Y., WERMAN M.: Representing local motion as a probability distribution matrix applied to object tracking. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (1997). 3, 4

[SBK10] SUNDARAM N., BROX T., KEUTZER K.: Dense point trajectories by gpu-accelerated large displacement optical flow. *European Conference on Computer Vision* (2010). 2

[SRB10] SUN D., ROTH S., BLACK M.: Secrets of optical flow estimation and their principles. In *IEEE Conference on Computer Vision and Pattern Recongition* (2010). 5, 6, 7

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision* (1998). 3

[WJW05] WAINWRIGHT M., JAAKKOLA T., WILLSKY A.: MAP estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory 51*, 11 (2005). 2

[WPB10] WERLBERGER M., POCK T., BISCHOF H.: Motion estimation with non-local total variation regularization. In *Proceedings of the conference on Computer Vision and Pattern Recognition* (2010). 2

[XCS*06] XIAO J., CHENG H., SAWHNEY H., RAO C., ISNARDI M.: Bilateral filtering-based optical flow estimation with occlusion detection. In *Proceedings of the European Conference on Computer Vision* (2006). 3

[YK06] YOON K.-J., KWEON I. S.: Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 4 (2006). 3

[YTA09] YANG Q., TAN K.-H., AHUJA N.: Real-time O(1) bilateral filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2009). 3