Fast Edge-Preserving PatchMatch for Large Displacement Optical Flow

Linchao Bao, *Student Member, IEEE*, Qingxiong Yang*, *Member, IEEE*, and Hailin Jin, *Member, IEEE*

***Abstract*—The speed of optical flow algorithm is crucial for many video editing tasks such as slow motion synthesis, selection propagation, tone adjustment propagation, etc. Variational coarse-to-fine optical flow algorithms can generally produce high-quality results but cannot fulfil the speed requirement of many practical applications. Besides, large motions in real-world videos also pose a difficult problem to coarse-to-fine variational approaches. We in this paper present a fast optical flow algorithm that can handle large displacement motions. Our algorithm is inspired by recent successes of local methods in visual correspondence searching as well as approximate nearest neighbor field algorithms. The main novelty is a fast randomized edge-preserving approximate nearest neighbor field algorithm which propagates self-similarity patterns in addition to offsets. Experimental results on public optical flow benchmarks show that our method is significantly faster than state-of-the-art methods without compromising on quality, especially when scenes contain large motions. Finally, we show some demo applications by applying our technique into real-world video editing tasks.**

I. Introduction

Optical flow estimation is one of the most fundamental problems in Computer Vision. Since the seminal work of Horn-Schunck *global* model [1] and Lucas-Kanade *local* model [2], there have been tremendous progresses in this area. We have algorithms that can handle challenging issues such as occlusions, motion discontinuities, textureless regions, etc. However, there are still outstanding problems in existing optical flow methods, such as large displacement motions and motion blur. This paper addresses the issue of large displacement motions. In particular, we are interested in fast optical flow algorithms as speed is crucial for practical applications.

Large displacement motions have been an issue in optical flow estimation since the beginning. The basic formulation of optical flow is based on a differential form of the brightness constancy equation which is invalid for motions larger than the support of the differential operators. In order to handle larger motions, traditional methods resort to the multi-scale coarse-to-fine framework. However, the coarse-to-fine framework suffers from an intrinsic limitation that it fails for fine scale image structures whose motions are larger than their size. Recently, there are several algorithms proposed to overcome this intrinsic limitation by going beyond the basic differential formulation and incorporating additional correspondence information. For instance, one can directly search for pixel correspondence [3]. But the complexity of the search step scales quadratically with respect to the size of the motion. Robust keypoints are one reliable source of correspondence information that can be matched efficiently across entire images but are only available at sparse image locations. Recently, an algorithm called deep-matching [4] is proposed to produce dense correspondence field efficiently, but its huge memory consumption prevents itself from practical applications. Besides, in order to obtain a dense flow field, one needs a global optimization step which is typically computationally expensive [5], [6].

In this paper, we propose to use approximate nearest neighbor field (NNF) for large displacement optical flow estimation. NNF is a correspondence field indicating pairs of image patches from two images which are closest in terms of some patch distance. There is no limitation on the relative distance between a pair of closest patches which makes NNF a good source of information for handling large displacement motions. Moreover, although exact NNF is expensive to compute, there exist efficient approximate algorithms [7], [8], [9].

In order to obtain optical flow using approximate NNFs, we need to address two fundamental problems. First, there is no spatial smoothness in a NNF which means neighboring patches in one image can have arbitrary matching patches in the other image. This problem is more pronounced in homogeneous regions where matching is ambiguous. Thus most approximate NNF algorithms (such as CSH [8] and KD-Tree [9] based algorithms) will produce messy fields and are not suitable for optical flow estimation. Second, occlusions are not respected in NNF computation, *i.e.*, one will get matching patches in occluded regions even though they are meaningless. The second problem can be resolved by explicitly performing consistency check between forward and backward flow. To address the first problem, one may attempt to use global optimization to incorporate motion candidates from a NNF into an optical flow estimation [10]. However, doing so may lead to a computationally expensive algorithm which has limited practical applicability. Instead, motivated by recent successes of local methods in stereo matching and optical flow [11], [12], [13] where it is shown that carefully crafted local methods can reach quality on par with global ones, we address the problem by increasing the local matching support (patch size). But increasing patch size leads to two new problems

Manuscript received May 22, 2014; revised July 27, 2014; accepted September 15, 2014. This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 21201914). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Dimitrios Tzovaras. (*Corresponding author: Q. Yang*)

L. Bao and Q. Yang are with the Department of Computer Science at City University of Hong Kong, Hong Kong S.A.R., China (e-mail: linchaobao@gmail.com; qiyang@cityu.edu.hk).

H. Jin is with Adobe Research, 345 Park Avenue, San Jose, CA 95110 (e-mail: hljin@adobe.com).

which are motion boundary preservation and algorithm speed. We address the former problem by introducing a novel edge-preserving version of PatchMatch [7] and the latter one by developing a fast approximate algorithm.

This paper extends its conference version [14] with the following major differences:

1) We provide more explanation of our method (see Section II), which is omitted in the conference version due to page limit.
2) We reveal more details about our implementation and experimental results in Section III.
3) We extend our method with a plane fitting scheme (see Section III-B) and add the details of the performance of our method on KITTI benchmark.
4) We apply our method to several real-world applications and show some examples in Section IV.

A. Related work

It is beyond the scope of this paper to review the entire literature on optical flow. Instead, we will only discuss the closely related papers. In particular, we will focus on the work that addresses large displacement motions. The classical coarse-to-fine framework for large displacement motions that is used by most optical flow algorithms was proposed in [15], [16] and refined in [17]. It generally works well for relatively large objects but performs poorly on fine scale image structures which may disappear in coarse scales. This is an intrinsic limitation of the coarse-to-fine framework. To overcome this limitation, Steinbruecker *et al.* [3] proposed to incorporate correspondence searches in a framework that avoids warping and linearization. However, the search part is exhaustive for every pixel in the image which makes the algorithm potentially slow for large search ranges. Instead of an exhaustive search at every pixel, the LDOF framework [5] is to only consider robust keypoints which serve as constraints in an energy-based formulation. Because keypoints can be matched across entire images, the algorithm does not suffer from the search range problem. To further improve the reliability of keypoint matching, the MDP-Flow [6] incorporated a discrete optimization step before diving into the variational optical flow solver.

Regarding NNFs, PatchMatch [7] was a seminal work and generated a lot of interests recently because of its computational efficiency and ability to match patches across large distance. But most algorithms in this area are proposed for the NNF problem in terms of reconstruction error [8], [9], which is different from the dense correspondence problem. Exceptionally, the work [18] applied PatchMatch to stereo matching for computing aggregation with slanted support windows, but they did not address the computational efficiency after adopting a weighting scheme on the support windows. A recent work employing NNF for optical flow estimation is [10], which computes an initial noisy but dense matching which is cleaned up through motion segmentation.

Our algorithm is closely related to the *local* methods in stereo matching and optical flow. Local methods have a long history in stereo matching. They used to be known as fast but less accurate compared to globally optimized methods. But [13] showed that a good local method can perform equally well. Rhemann *et al.* [11] successfully applied this principle to optical flow and obtained an algorithm that ranks high on the Middlebury flow benchmark. The SimpleFlow [12] followed the same direction but towards a less accurate yet faster solution. The PatchMatch Filter [19] adapted the PatchMatch algorithm onto superpixels and employed the algorithm from [11] to refine the matching correspondence between superpixels.

B. Contributions

The main contribution of this work is a fast local optical flow algorithm that can handle large displacement motions. Our method is local, *i.e.*, it does not involve optimization over the entire image and therefore fast. On the other hand, our method does not sacrifice on quality. We compare our method against existing ones on MPI Sintel, KITTI, and Middlebury benchmarks. Our ability to handle large displacement motions is clearly demonstrated by the top performance on the MPI Sintel benchmark. In terms of quality, our method outperforms all other fast methods without compromising on the speed. In fact, the quality of our method is on par with that of global ones but the speed is significantly faster.

Our main technical novelty is a fast randomized edge-preserving approximate nearest neighbor field algorithm. The key insight is that in addition to similar offsets, neighboring patches have similar self-similarity patterns. Therefore, we can propagate self-similarity patterns in a way similar to propagating offsets as done in [7]. This significantly reduces the computational complexity. We hope this idea to inspire other work in generalizing [7] to other applications.

II. Our Approach

Our method follows the traditional local correspondence searching framework [20] which consists of 4 steps:

1) matching cost computation,
2) cost aggregation,
3) correspondence selection, and
4) refinement.

It is shown that the framework can produce high-quality optical flow [11], but its computational complexity is linear in search range.

While reducing the correspondence search range may be a potential solution, we in this paper address this problem from another point of view. We notice that, if we use squared error as the matching cost and use box filtering to perform the cost aggregation, then steps (1) to (3) are actually equivalent to searching the nearest neighbors for image patches using the patch Euclidean distance, which is known to have fast approximate algorithms that are independent of search range, such as PatchMatch [7]. However, a direct use of PatchMatch to estimate optical flow can handle large displacement motions but tend to introduce visible errors around motion discontinuities as shown in Fig. 1b. To overcome the problems, we propose a new edge-preserving version of PatchMatch (Sec. II-A) and a corresponding fast algorithm (Sec. II-B).

The techniques used in [11] for the refinement step (*i.e.*, consistency check and weighted median filtering [21], [22]) are also employed in this paper except that we suggest to produce subpixel accuracy with a more efficient technique – paraboloid fitting, which is a 2D extension from the 1D parabola fitting – a commonly adopted technique in stereo matching [23]. Details are presented in Sec. II-C and II-D.

A. Edge-Preserving PatchMatch

The main idea of original PatchMatch [7] is to initialize a random correspondence field and then iteratively propagate good guesses among neighboring pixels. In order to avoid trapping into local minima, several random guesses are additionally tried for each pixel during the propagation. The matching cost between two patches is originally defined as the patch Euclidean distance. Specifically, suppose two patches with radius r are centered at location $\mathbf{a}(x_a, y_a)$ in image A and location $\mathbf{b}(x_b, y_b)$ in image B, respectively. The matching cost between the two patches is

$$d(\mathbf{a},\mathbf{b}) = \sum_{\substack{\mathbf{\Delta}(\Delta x,\Delta y):\\ |\Delta x|\leqslant r, |\Delta y|\leqslant r}} \|I^A(\mathbf{a}+\mathbf{\Delta}) - I^B(\mathbf{b}+\mathbf{\Delta})\|^2, \quad (1)$$

where I^A and I^B denote the CIELab color appearances of image A and B, respectively.

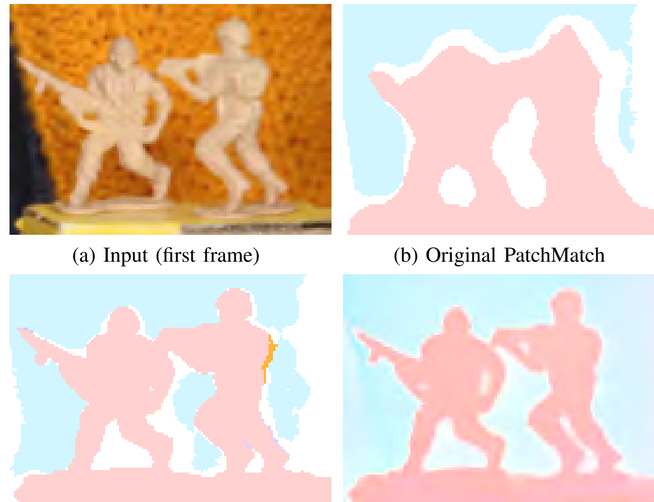
In order to make the NNF preserve details of input image, we add *bilateral weights* [13] into the matching cost calculation . Moreover, similar to the data term employed in variational optical flow estimation [24], [25], we replace the L_2 norm in the above formulation with a robust loss function (such as the negative Gauss function or the Lorentzian function [24]) to reject outliers. Furthermore, in addition to color cue, we can add more cues that can better deal with repetitive patterns and textureless regions, *e.g.*, image gradient or the census transform [26]. Specifically, the matching cost in our approach is defined as follows,

$$d(\mathbf{a},\mathbf{b}) = \frac{1}{W} \sum_{\substack{\mathbf{\Delta}(\Delta x,\Delta y):\\ |\Delta x|\leqslant r, |\Delta y|\leqslant r}} w(\mathbf{a},\mathbf{b},\mathbf{\Delta}) C(\mathbf{a},\mathbf{b},\mathbf{\Delta}), \quad (2)$$

where $w(\cdot)$ is the bilateral weighting function, W is the normalization factor (sum of all the weight $w(\cdot)$), and $C(\cdot)$ is the *robust* cost between two pixels (suppose K cues are involved in the cost calculation):

$$\begin{aligned} w(\mathbf{a},\mathbf{b},\mathbf{\Delta}) &= \exp(-\frac{\|I^A(\mathbf{a}+\mathbf{\Delta}) - I^A(\mathbf{a})\|^2}{\sigma_r^2}) \\ &\quad \exp(-\frac{\|I^B(\mathbf{b}+\mathbf{\Delta}) - I^B(\mathbf{b})\|^2}{\sigma_r^2}) \\ &\quad \exp(-\frac{\|\mathbf{\Delta}\|^2}{\sigma_s^2}), \quad (3) \\ C(\mathbf{a},\mathbf{b},\mathbf{\Delta}) &= \sum_{i=1}^{K} \rho_i(C_i^A(\mathbf{a}+\mathbf{\Delta}) - C_i^B(\mathbf{b}+\mathbf{\Delta})), \quad (4) \end{aligned}$$

where σ_s and σ_r are controlling spatial and range influences, respectively (typically, we set $\sigma_s = 0.5r$ (r is patch radius) and $\sigma_r = 0.1$. The cost contributed by each cue C_i is controlled by a robust loss function $\rho_i(\cdot)$ for rejecting outliers and balancing between different cues (for simplicity, we use the same loss function for all the cues used in our experiments, see Sec. III).



(a) Input (first frame) (b) Original PatchMatch (c) Ours (d) Ours (refined)

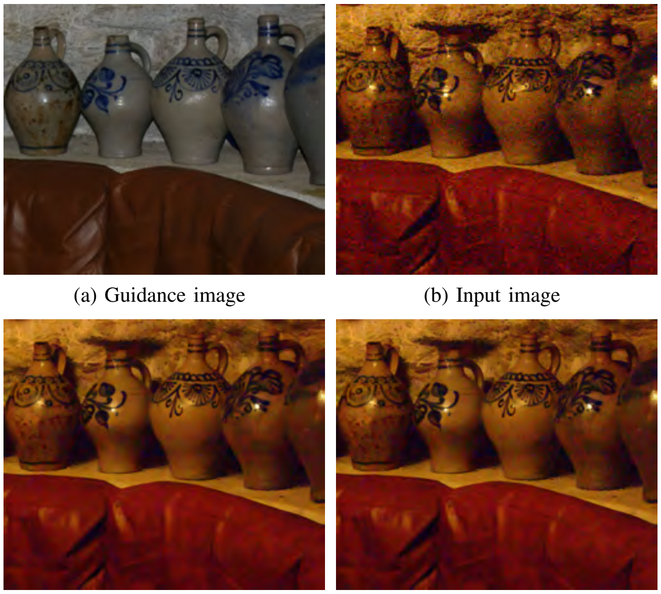
Fig. 1. PatchMatch results (cropped) on the "Army" dataset from Middlebury benchmark [27]. The proposed edge-preserving PatchMatch can preserve details in the NNF results. Note that the outliers in NNF result can be easily removed by refinement.

Fig. 1 shows a comparison of the NNF results produced by the original PatchMatch and the proposed edge-preserving PatchMatch. The details in input image can be much better preserved in NNF when using our edge-preserving version. Note that in order to perform flow refinement (in particular, the consistency check [11]), we need to compute the NNFs between two images in both directions. Thus we use the *symmetric* bilateral weight in Eq. (3), so that during the PatchMatch we can symmetrically update both NNFs after calculating one matching cost.

B. Approximate Algorithm

While PatchMatch can effectively reduce computational complexity in terms of search range, its complexity still depends on patch size. In order to produce high-quality flow fields, however, a large patch size is usually preferred for eliminating matching ambiguities (note that the edge-preserving feature plays an important role for maintaining flow accuracy when increasing patch size). In this section, we propose an algorithm that utilizes a self-similarity propagation scheme and a hierarchical matching scheme to approximate the exact edge-preserving PatchMatch.

1) Self-Similarity Propagation: We notice that, due to the range kernel employed in the matching cost computation (Eq. (3)), the major portion of the matching cost is contributed by pixels that are similar to the center pixel. This suggests a natural way to accelerate the matching cost computation which is to simply ignore dissimilar pixels to center pixel. To be more specific, for each pixel, we precompute the n ($n \ll M = (2r+1)^2$) most similar pixels within its neighborhood, store their positions and use them to compute the cost.



(a) Guidance image (b) Input image

(c) Brute-force JBF (d) $n = 100$ (38dB)

Fig. 2. Approximating joint bilateral filtering (JBF) using n most similar pixels for each pixel ($r = 17$, patch size is 35×35, $\sigma_s = 0.5 \times r$, $\sigma_r = 0.05$). Input images and parameters are reproduced from [28]. It is suggested that PSNR value above 40dB often corresponds to almost invisible differences between two images [30].

Average Error	EPE	AAE	CPU Timing (sec)[†]
Patch (35×35)	0.31	3.35	97.8
$n = 200$	0.32	3.45	19.1
$n = 100$	0.33	3.50	10.2
$\mathbf{n = 50}$	**0.33**	**3.56**	**5.4**
$n = 30$	0.49	5.08	3.5
$n = 10$	0.91	9.94	1.6

[†]The time is recorded for running bidirection PatchMatch algorithm (computing two NNFs) on 640×480 images. Accuracy is evaluated after refinement. Note that the CostFilter [11] takes about 430 seconds on the same CPU to produce bidirectional optical flow with search range 61×61.

TABLE I
AVERAGE OPTICAL FLOW ACCURACY ON THE MIDDLEBURY TRAINING DATASETS WHEN USING SELECTED PIXELS (THE n MOST SIMILAR PIXELS FOR EACH PIXEL) TO COMPUTE MATCHING COST.

Before applying the idea to optical flow estimation, we first performed experiments on joint bilateral filtering [28], [29] to validate it, since the matching cost computation in Eq. (2) is *essentially* performing a brute-force joint bilateral filtering on cost cues (using input image as guidance). The experiment is conducted as follows: we compute the output of each pixel only by the n most similar pixels to it according to the guidance image. It is shown in the experiment that $n = 50$ to 100 can commonly produce high-quality approximate results for large patch size like 35×35, which is employed in our optical flow algorithm. Fig. 2 shows one example of the experimental results.

When it comes to optical flow estimation, we performed experiments on the Middlebury training datasets [27] to validate this idea. For each pixel, the neighboring n most similar pixels are used for computing patch matching cost. Table I shows the optical flow accuracy and the corresponding runtime on Middlebury training datasets when n is with different value

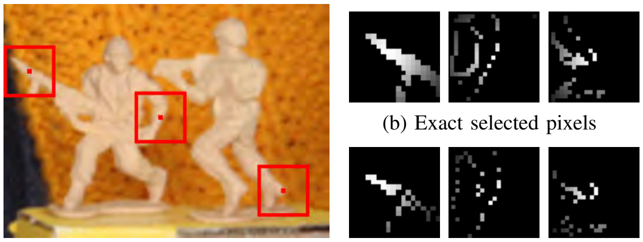
Algorithm 1 Self-Similarity Propagation Algorithm

Input: image A, patch radius r, number of selected pixels n.
Output: a self-similarity vector $S(x, y)$ for each pixel (x, y) containing locations of n selected pixels.
/* Initialization */
for each pixel (x, y) in A **do**
(1) randomize a vector $S(x, y)$ containing the locations of n neighboring pixels within the patch centered at (x, y);
(2) sort pixels in $S(x, y)$ according to their Euclidean similarity in CIELab color space to center pixel (x, y).
end for
/* Propagation */
for each pixel (x, y) in A (scan from top-left to bottom-right) **do**
(1) merge vector $S(x-1, y)$ and $S(x, y-1)$ into $S(x, y)$ according to the pixels' color similarity to pixel (x, y);
/ This is essentially a sorting over $3n$ pixels in the three vectors and select the n best pixels according to their color distances to pixel (x, y). However, when pixel color at $(x-1, y)$ or $(x, y-1)$ is close to (x, y), the merge process can be done faster with a process similar to a merge sorting since the three vectors can be treated as sorted vectors. */*
(2) for pixel in $S(x, y)$ that falls outside the patch window, randomize a new location within the patch and re-sort the vector.
end for
for each pixel (x, y) in A (scan from bottom-right to top-left) **do**
(1) merge vector $S(x+1, y)$ and $S(x, y+1)$ into $S(x, y)$ (similar to above);
(2) for pixel in $S(x, y)$ that falls outside the patch window, randomize a new location within the patch and re-sort the vector.
end for

(patch size is fixed to 35×35). Surprisingly, the result gives a very good support for applying this idea to optical flow estimation – upon balancing between quality and efficiency, $n = 50$ can be a very good choice for 35×35 patch, which is much smaller than the number of pixels inside each patch. By involving much less pixels when computing matching cost, the runtime of PatchMatch algorithm can be substantially reduced, while only sacrificing very little quality performance.

Then a problem raised is that the brute-force selection of n most similar pixels for each center pixel (out of total M pixels within a patch) actually can be too slow, especially when patch size is large, which may cancel out a large portion of the speed gain of using less pixels. For example, selecting $n = 50$ out of 35×35-sized patch for 640×480 image takes about **12** seconds in our experiments (on CPU). Note that the straightforward implementation of the selection process takes $O(Mn)$ complexity for each pixel. With a complex data structure (like a max-heap), the computation complexity can only be reduced to $O(M \log n)$, which is still too high. Fortunately, inspired by the spirit of PatchMatch itself, we designed a self-similarity propagation algorithm to roughly select similar pixels for each pixel in a much faster way.

Our self-similarity propagation algorithm utilizes the fact that adjacent pixels tend to be similar to each other, just like the PatchMatch itself. Specifically, the algorithm is as follows: for each pixel, we randomly select n pixels from its surrounding region and store them into a vector in the order of their similarity to the center pixel (namely, self-similarity vector); then we scan the image from top-left to bottom-right, and, for each pixel, merge its adjacent pixels'



(a) Input (b) Exact selected pixels (c) Approximated

Fig. 3. Visual results of our approximated algorithm for selecting similar pixels ($n = 50$). The approximate results appear to have less pixels because of duplicated pixel selection.

vector into its own vector (according to the stored pixels' similarity to current pixel); reversely scan and merge. Since we do not intend to select exactly the n most similar pixels to the center pixel for each patch, the algorithm does not need to interleave additional random guesses during propagation or iterate more. The pseudo-code is in Algorithm 1. The approximate algorithm only needs $O(n \log n)$ computation for each pixel (the sorting in initialization step and merging in propagation step), which is independent of patch size. Thanks to the propagation between adjacent pixels, the algorithm can produce reasonably good approximate results in a much faster speed (for 35×35 patch size with $n = 50$, it takes about **1.8** seconds and is about **6x** faster than the exact selection, the speedup factor grows larger as the patch size becomes larger). For the optical flow accuracy, we do not experience degraded accuracy on the Middlebury training datasets than using the exact selected n pixels as reported in Table I. Fig. 3 shows an example of the visual results of the selected pixels by our algorithm. More results are provided in Sec. III.

2) Hierarchical Matching: When input image is large, performing PatchMatch on all pixels is a waste of computation. We employ a hierarchical matching scheme to further accelerate the algorithm. Specifically, given a pair of input frames, we first downsample the images to a certain lower resolution (for a balance between speed and accuracy, we typically downsample input images twice with a factor 0.5 at each dimension), then we perform the above algorithm to compute the NNF on the downsampled images. After obtaining the NNF on lower resolution, we perform joint bilateral upsampling [23] to get a coarse NNF on higher resolution. Then we perform a 3×3 local patch matching to refine the coarse NNF on the higher resolution images. The pipeline is repeated until we finally get the NNF on the original resolution.

The hierarchical scheme is somewhat similar to that was used in SimpleFlow [12]. However, there are two key differences between our approach and theirs: first, since our edge-preserving PatchMatch does not have restriction on search range, we do not downsample the original frames to very low resolutions and hence it is able to handle large displacements of thin structures (if they still exist in the downsampled resolution). This will also reduce large error accumulation when propagating NNF estimate from much lower resolution to higher resolution. Second, thanks to the edge-preserving ability, the coarser NNF is usually accurate enough and we only need to perform local search within a 3×3 neighborhood when refining the NNF on higher resolution. This can largely reduce the computation cost, thus our approach is much faster than SimpleFlow (see Sec. III).

Notice that although the hierarchical matching scheme here is similar to the traditional coarse-to-fine framework, the early-cutting pyramid employed in our method is essential to make our method effective for handling large displacement (see Sec. III for the experimental validation for handling large displacement motions).

C. Handling Occlusions and Outliers

After computing bidirectional NNFs (at each resolution) between two images, we explicitly perform forward-backward consistency check [11] between the two NNFs to detect occluded regions. Inconsistent mapping pixel is then fixed by nearby pixels according to their bilateral weights. Even so, there will still be some incorrect mapping pixels that cannot be detected by the consistency check, which we treat as *outliers*. A weighted median filtering [31] is thus performed on the flow fields to remove the outliers (filtering is performed on *all* pixels). A second pass consistency check and fixing is then performed to make sure the filtering does not introduce inconsistency. Note that the consistency check and fixing is usually very fast, the computational overhead in this step is mainly the weighted median filtering performed on all pixels.

When occluded region is large, a dedicated hole-filling step is needed in order to fill flow values into such region (*e.g.*, in our implementation a simple scanline-based algorithm is used). Notice that in practical applications, occluded region is not necessarily to be filled with flow values. On the contrary, it is actually a reliable way to detect occluded regions and choose specific handling algorithm with respect to different applications (e.g., the application in Sec. IV-A).

D. Subpixel Refinement

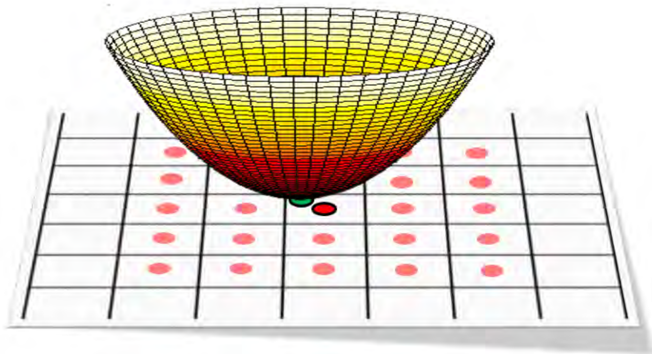
Suppose the discrete correspondence for each pixel $\mathbf{a}$ in image A is $NN_{A\to B}(\mathbf{a}) = \mathbf{b}$, and the patch centered at pixel $\mathbf{a}$ is denoted by Ω_a. We then compute the matching costs between patch Ω_a and m different patches around patch Ω_b (see Fig. 4a), respectively, which is denoted as $\mathcal{D} = \{d_1, d_2, ..., d_m\}$. Note that when computing the matching cost, the fast algorithm in previous section still applies. Assume the cost follows a paraboloid surface on the 2D image grid:

$$d = f(x, y) = \theta \cdot [x^2, y^2, xy, x, y, 1]^{\mathrm{T}}, \tag{5}$$

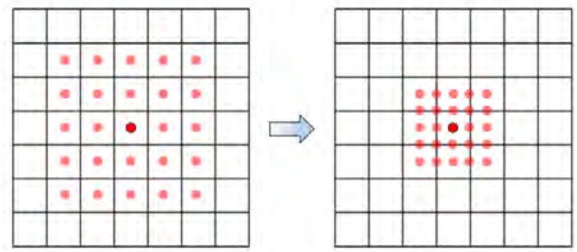
where $\theta = [\theta_1, \theta_2, ..., \theta_6]$ are the unknowns. Substituting the m ($m \geqslant 6$, typically 25 in our experiments) known points into the equation, we can solve the linear system and figure out the unknowns. Then the $\mathbf{b}^*(x^*, y^*)$ associated with the minimum cost can be computed as follows (by taking derivatives and setting them to zero),

$$x^* = \frac{2\theta_2\theta_4 - \theta_3\theta_5}{\theta_3^2 - 4\theta_1\theta_2}, \quad \text{and} \quad y^* = \frac{2\theta_1\theta_5 - \theta_3\theta_4}{\theta_3^2 - 4\theta_1\theta_2}, \tag{6}$$

which is the location of $\mathbf{a}$'s correspondence with subpixel accuracy. Note that the linear system to be solved is very small,

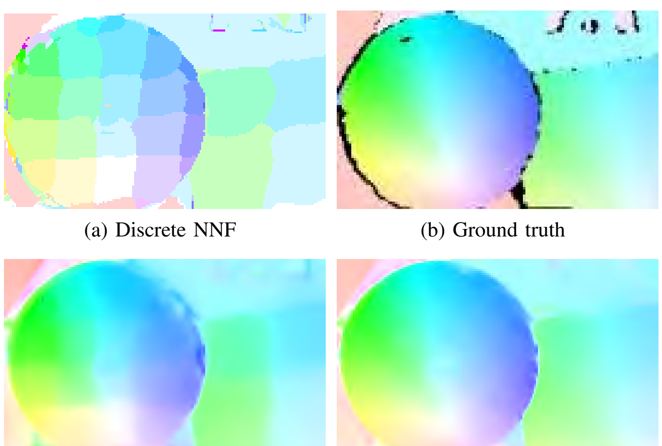


(a) Paraboloid fitting



(b) Improved

Fig. 4. Illustration of our subpixel refinement. In (a): the red circles stand for m ($m = 25$ in the figure) patch centers around the original target position. An elliptic paraboloid is fitted using the m points with their patch matching costs and the resulting subpixel position (marked by green circle) is computed as the bottom vertex of the paraboloid. In (b): the m patches from upsampled images are actually denser on the original resolution and the resulting subpixel position is more accurate. Note that the improved result is computed with almost no additional cost (except for pixel value interpolation, which is negligible compared to other computation).



(a) Discrete NNF (b) Ground truth

(c) Subpixel refined (d) Improved

Fig. 5. Example of subpixel refinement. Note that the improved result in (d) is obtained in the *same* runtime as that in (c).

in practice if we multiply a transposed matrix on both sides, the linear system will have a constant size of 6×6, no matter how many points are involved (the value of m).

To further increase the subpixel accuracy, we compute matching cost for the m points on upsampled images instead of the original images (we obtain upsampled image using bicubic interpolation with an upsampling factor of 2 along each dimension in all our experiments). This does not increase the computational overhead since we only need to compute matching cost for all pixels on the original resolution. The main difference is that the m points around pixel $\mathbf{b}$ are now already with subpixel offsets to $\mathbf{b}$ (see Fig. 4b). Fig. 5 shows the improvement of this strategy.

Finally, an edge-preserving filtering with small parameters (*e.g.*, bilateral filtering [32], [33] with $\sigma_s = 2, \sigma_r = 0.01$ in our experiments) is performed on the flow fields to smooth out small outliers that might be introduced in this step.

Clean pass	EPE all	EPE s40+†	Runtime (sec)‡	Processor
DeepFlow[4]	5.377	33.701	17	CPU
MDP-Flow2[6]	5.837	39.459	547	CPU
Ours	6.494	39.152	**0.25**	GTX 780
S2D-Match[34]	6.510	44.187	1920	CPU
Classic+nlp[25]	6.731	45.290	888	CPU
FC-2Layers[35]	6.781	45.962	4525	CPU
LDOF[5], [36]	7.563	51.696	2.7	GTX 580
Classic+nl[25]	7.961	57.374	888	CPU
Classic++[25]	8.721	60.645	510	CPU
Horn-schunck[1]	8.739	58.243	156	CPU
Cls+nl-fast[25]	9.129	66.935	174	CPU
SimpleFlow[12]	12.617	81.786	2.9	GTX 285
A.Huber-L1[37]	12.642	77.835	3.2	GTX 280

Final pass	EPE all	EPE s40+†	Runtime (sec)‡	Processor
DeepFlow[4]	7.212	44.118	17	CPU
S2D-Match[34]	7.872	48.782	1920	CPU
FC-2Layers[35]	8.137	51.349	4525	CPU
Classic+nlp[25]	8.291	51.162	888	CPU
Ours	8.377	49.083	**0.25**	GTX 780
MDP-Flow2[6]	8.445	50.507	547	CPU
LDOF[5]	9.116	57.296	2.7	GTX 580
Classic+nl[25]	9.153	60.291	888	CPU
Horn-schunck[1]	9.610	58.274	156	CPU
Classic++[25]	9.959	64.135	510	CPU
Cls+nl-fast[25]	10.088	67.801	174	CPU
A.Huber-L1[37]	11.927	74.796	3.2	GTX 280
SimpleFlow[12]	13.364	81.350	2.9	GTX 285

†: The column "EPE s40+" means the average endpoint error over regions with flow velocities larger than 40 pixels per frame.

‡: The runtime are reproduced from either the original papers or the other benchmark websites for 1024×436 sized images. Note that due to large memory consumption, DeepFlow [4] is difficult to be implemented on GPU.

TABLE II
PERFORMANCE ON MPI SINTEL BENCHMARK (SINTEL.IS.TUE.MPG.DE/RESULTS, CAPTURED ON MAY 15, 2014). ONLY PUBLISHED PUBLICATIONS ARE SHOWN.

III. EXPERIMENTAL RESULTS

In this section, we present our experimental results on three public optical flow benchmarks – the Middlebury benchmark [27], the KITTI benchmark [38], and the MPI Sintel benchmark [39]. Note that the Middlebury benchmark only contains

(a) Input (first frame) (b) Input (second frame) (c) Ground truth (d) Our result

Fig. 6. Example results on MPI Sintel training data. The scenes are pretty challenging because of large motions and motion blur. Notice that most of the motion boundaries are well preserved in our results.

(a) "ambush_1" clean pass (b) "ambush_1" final pass

(c) "ambush_3" clean pass (d) "ambush_3" final pass

Fig. 7. Visual comparison of our results on the two passes of MPI Sintel benchmark. The heterogeneous smoke in (b), as well as the "textured" fog (see Fig. 8), seriously disturbs image local variances and cause the results of our *local* method degraded much, especially at textureless regions (see the regions marked by red squares).

small displacement motions and the KITTI benchmark is specially targeted on autonomous driving, thus our main focus is on the MPI Sintel benchmark. In our implementation, we use the AD-Census [40] for computing matching cost (*i.e.*, the CIELab color cue together with the census transform cue). Parameters for the edge-preserving PatchMatch are set to $r = 17$, $\sigma_s = 0.2r$ and $\sigma_r = 0.1$. We implemented the whole pipeline of our algorithm using CUDA and performed all the experiments on a NVIDIA Geforce GTX 780 GPU. In the self-similarity propagation step, We employed the Jump Flooding algorithm [41] for fast parallel propagation on GPU (the same as PatchMatch [7]).

A. Results on MPI Sintel Benchmark

The MPI Sintel benchmark is a challenging optical flow evaluation benchmark, especially due to the complex elements

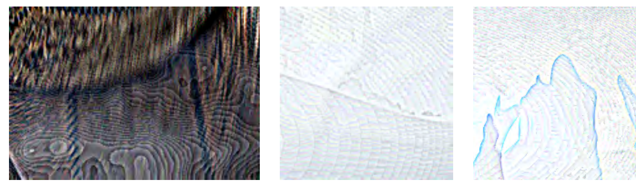


Fig. 8. Close-ups (marked by red squares in Fig. 7) for the detail enhanced version of the input images in Fig. 7. The subtle textures introduced by synthetic atmospheric effects can be easily observed.

involved, *e.g.*, large motions, specular reflections, motion blur, defocus blur, and atmospheric effects. The evaluation is performed on two kinds of rendering frames, namely *clean pass* and *final pass*, each containing 12 sequences with over 500 frames in total. Table II shows the performance of our method on this benchmark (complete table is available online). Our method are among the top performers but with much faster speed than the competitors. Note that if we only consider regions containing large motions (see column "EPE s40+" in Table II), our method ranks even higher. Fig. 6 shows two examples of our results on the training data.

One observation is that our method performs worse on the final pass than on the clean pass. Note that the final pass is rendered with motion blur, defocus blur and atmospheric effects while the clean pass are not. By comparing between the results on the two passes (see Fig. 7), we find that our results are mainly degraded on 3 (out of 12) sequences, namely "ambush_1", "ambush_3", and "mountain_2," when moving from clean pass to final pass. In fact, it turns out motion blur and defocus blur do not affect the quality of the results too much, since adjacent frames are usually blurred similarly. This is also usually true for real-world videos, except when the observed object dramatically changes speed or the camera changes focus. The real reason why the results are degraded on the 3 sequences is actually because of the synthetic atmospheric effects, in particular, the heterogeneous smoke (Fig. 7b) and heavy fog (Figs. 7d). These two kinds of effects seriously disturb image local variances (while this is obvious for smoke, the synthetic fog actually introduces very subtle textures, which can be observed on the detail enhanced input images shown in Fig. 8), and this will cause problems at textureless regions for local method since matching cues

Method†	Out-Noc	Avg-Noc	Runtime (sec)‡	Processor
TGV2adcsift[42]	4.71%	1.6px	12	GTX 460
DeepFlow[4]	5.38%	1.5px	17	CPU
CRTflow[43]	6.90%	2.7px	18	GTX 460
Classic++[44]	8.04%	2.6px	510	CPU
fSGM[45]	8.44%	3.2px	60	CPU
Ours	8.62%	2.5px	**0.25**	GTX 780
TGV2census[46]	9.19%	2.9px	4	GTX 460
Cls+nl-fast[44]	10.13%	3.2px	174	CPU
Ours (w/o PF)	12.28%	3.4px	**0.23**	GTX 780
Horn-schunck[1]	12.47%	4.0px	156	CPU
LDOF[5]	18.72%	5.5px	2.4	GTX 580
TV-L1[47]	26.50%	7.8px	16	CPU
BERLOF[48]	30.63%	8.5px	0.231	GTX 680
RLOF[49]	31.49%	8.7px	0.488	GTX 680
HAOF[50]	32.48%	11.1px	16.2	CPU
PolyExpand[51]	44.53%	17.2px	1	CPU
Pyramid-LK[2]	57.22%	21.7px	90	CPU

†: Only pure optical flow algorithms are shown. Methods incorporated with stereo matching or epipolar geometry are not shown.

‡: The runtime are reproduced from either the original papers or the other benchmark websites for 1242×375 sized images.

TABLE III
PERFORMANCE ON KITTI BENCHMARK (WWW.CVLIBS.NET/DATASETS/KITTI, CAPTURED ON MAY 15, 2014) WHEN ERROR THRESHOLD IS 5 PIXEL. "OURS (W/O PF)" IS OUR METHOD WITHOUT PLANE FITTING SCHEME.

might be locally dominated by the subtle textures introduced. See Sec. III-D for more discussion.

B. Results on KITTI Benchmark

The KITTI optical flow benchmark contains 194 pairs of grayscale frames (test dataset), which are obtained with a wide-view camera fixed on a moving vehicle. Thus most of the scenes in the dataset are perspective views of streets and the scene motions are caused by camera movements along streets. In this case, the frontal-parallel assumption of PatchMatch often fails due to the slanted planes (*e.g.*, the road ahead of camera) in the scenes. Our method without plane fitting scheme performs not well on this benchmark (see "Ours (w/o PF)" in Table III).

In order to adjust our method to better handle such kind of scenes, we introduce the randomized plane fitting scheme into our method [18]. The idea is that, during the patch matching, the shape of the patch is adjusted to fit the optimal plane orientation. Since the optimal plane is unknown, it is parameterized with three unknown parameters for each pixel, which is initialized with random guess and propagated during PatchMatch (just like the unknown flow itself) [18]. The improvement of this scheme is particularly effective for KITTI benchmark (see the entry "Ours" in Table III). Fig. 10 shows a visual example of the improvement.

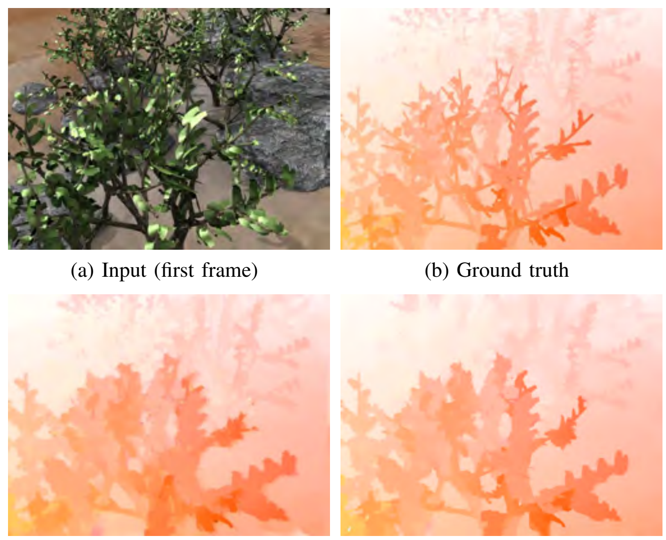
Notice that since most of the flow in KITTI benchmark are caused by camera movement and tend to be smooth with few motion boundaries, traditional coarse-to-fine methods (*e.g.*, "Classic++" [44]) actually perform better than some other more advanced methods (including ours). However, if one is willing to compromise a little on quality for the sake of speed, our method can provide a good choice in this case.

Method	Avg. Rank	Avg. EPE	Runtime (sec)†	Processor
Ours (w/o HM)	31.1	0.33	**2.5**	GTX 780
SimpleFlow[12]	35.5	0.47	1.7+240‡	GTX 280
Adaptive[52]	38.9	0.40	9.2	Tesla C1060
CompOF-FED[53]	43.0	0.47	0.97	GTX 285
A.Huber-L1[37]	44.2	0.40	2	GTX 285
TV-L1-Imp[47]	49.1	0.54	2.9	GTX 285
WTV-L1-SB[54]	51.0	0.48	0.38	GTX 780
Ours	64.0	0.62	**0.20**	GTX 780

†: The runtime are reproduced from the benchmark website (only algorithms reported on GPU are shown). Note that considering the hardware generations, CompOF-FED [53] actually should be faster than our method. But only our algorithm in this table can handle large displacement motions.

‡: The reported results of SimpleFlow are obtained after global optimization using [25], which takes about 240 seconds using the Matlab code provided by [25].

TABLE IV
PERFORMANCE ON MIDDLEBURY BENCHMARK (VISION.MIDDLEBURY.EDU/FLOW, ENDPOINT ERROR, CAPTURED ON MAY 15, 2014). "OURS (W/O HM)" IS OUR METHOD WITHOUT HIERARCHICAL MATCHING SCHEME.

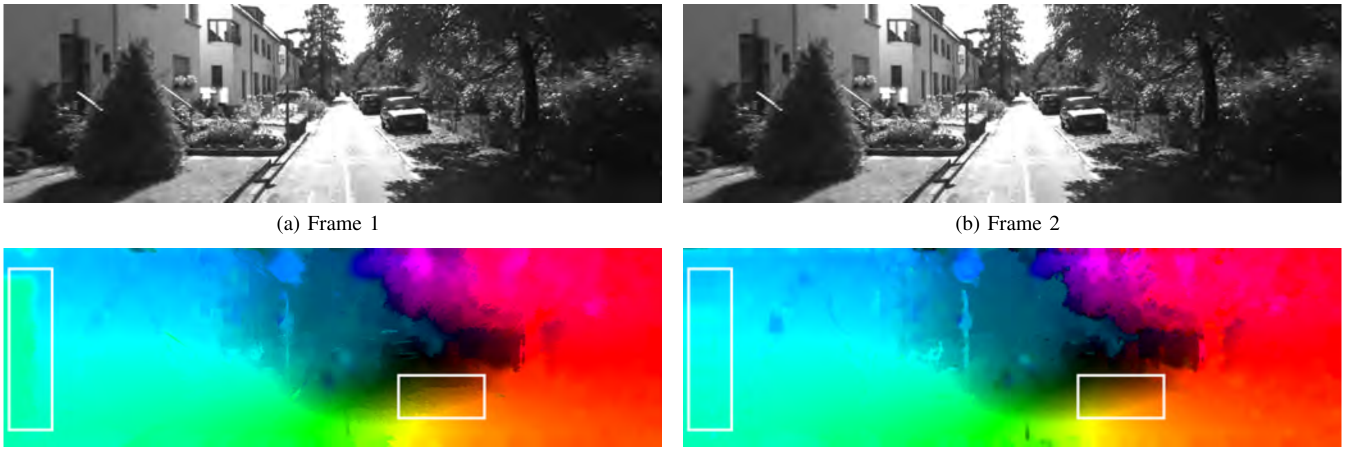


(a) Input (first frame) (b) Ground truth (c) with HM (0.82) (d) without HM (0.63)

Fig. 9. An example of our results on Middlebury benchmark. The EPE is shown in the caption.

C. Results on Middlebury Benchmark

The evaluation on Middlebury Benchmark is performed on 12 pairs of frames, most of which contain only small displacement motions. Since a matching process is not necessarily needed in the context of small displacements, our method is actually not suitable for this benchmark. Table IV shows the performance of our method on the Middlebury benchmark (complete table is available online). Note that since the evaluation dataset is very small, methods submitted to the benchmark tend to be overfitted (a small difference in EPE can lead to a huge difference in ranking). Our algorithm without the hierarchical matching scheme gets a large promotion on the ranking list (see "Ours (w/o HM)" in Table IV, notice that hierarchical matching scheme is for fast approximation).



(a) Frame 1 (b) Frame 2

(c) Without plane fitting (4.27%) (d) With plane fitting (3.67%)

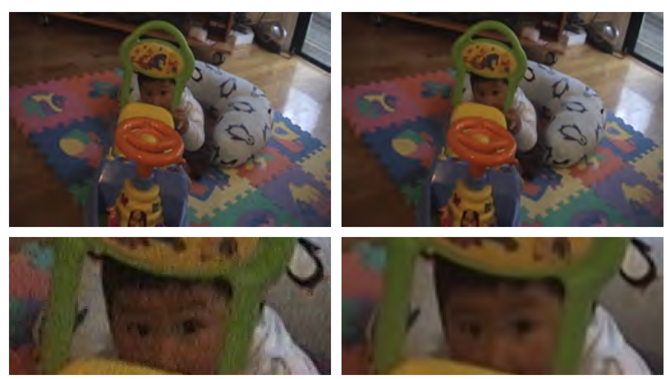
Fig. 10. The improvement by introducing plane fitting scheme. The color coding of flow is from KITTI benchmark. The number in the captions means percentage of bad pixels (flow error larger than 5 pixels). Notice the improvement in the region marked by the white squares.

(a) Input ($\Delta t = 0.0$) (b) Result ($\Delta t = 0.2$) (c) Result ($\Delta t = 0.4$) (d) Result ($\Delta t = 0.6$) (e) Result ($\Delta t = 0.8$) (f) Input ($\Delta t = 1.0$)

Fig. 11. Two snapshots of our result on video slow motion synthesis. Please refer to the supplementary materials for video examples.

D. Limitations

As a local method, our approach fails at large textureless regions, where local evidences are not enough to eliminate matching ambiguities. While increasing patch size or adding more cues (such as the census transform) might help relieve the problem, it cannot be completely avoided, especially when the regions are large. In addition, textureless regions can be easily affected by small noise or disturbance (such as the synthetic "textured" fog in Fig. 8), which may lead to incorrect match. In this case, global optimization techniques may be needed. However, notice that mismatch in textureless regions might not be a serious problem for some real-world applications.



(a) Input (b) Our result

Fig. 12. A snapshot of our results on flow-based video denoising. The original video and denoised video can be found in the supplementary materials.

IV. Demo Applications

We in this section show some results of applying our optical flow algorithm into real-world video editing applications. Notice that in these applications, a fast optical flow algorithm is the key to provide a user-friendly editing experience.

A. Video Slow Motion Synthesis

Slow motion synthesis is to construct artificial frames between existing video frames to make the video slow down to a certain speed. For example, if the video is going to be slow down to 10% speed, the number of total video frames should be 10x more and thus there should be 9 frames to be constructed between each two existing frames. The natural way to construct an artificial frame is to fetch pixels from the two existing frames between which the frame is to be placed. We compute the bidirectional (forward and backward) optical flow between the two existing frames, and then perform a forward warping on the flow fields to get two intermediate flow fields for a given time position between the two frames,

(a) First frame (user input) (b) Second frame (our result)

Fig. 13. A snapshot of our results on video selection propagation. The red line in the first frame is selected by user and that in the second frame is automatically produced using optical flow.

and then construct two intermediate artificial frames from the two existing frames, respectively. Finally we use the given time position to blend the two intermediate artificial frame to get a result frame, which is more similar to the frame to which the time position is closer. The optical flow computation is the most time-consuming part of this pipeline (the runtime of the other part can be ignored comparing to the runtime of optical flow algorithm). With our optical flow algorithm, we are able to achieve **40** fps (based on the number of output frames) for 640x480 videos when the target motion speed is 10%. Fig. 11 shows one example of our results. More video results can be found in the supplementary materials.

B. Flow-based Video Denoising

Optical flow can be used to establish temporal correspondences across video frames for high-quality video denoising [55]. For each patch in a certain frame, we first compute a few similar patches around it and then use the optical flow fields to collect more similar patches in nearby frames (according to the original patch and its similar neighbors). Finally, an algorithm similar to non-local means is applied to denoise the original patch using all the collected patches. The pipeline is pretty robust for producing temporal coherent video results. The main computation is again the optical flow estimation step. With our optical flow algorithm, the whole pipeline could be accelerated much. Fig. 12 shows an example of our results. More video results can be found in the supplementary materials.

C. Video Editing Propagation

A common way to reduce user efforts in video editing is to propagate user editing results (such as object selection, recolorization, and tone adjustment) from one frame to another according to optical flow field. Fig. 13 shows an example of propagating user selected object boundary with our flow result.

V. Conclusions

In this paper, we present an optical flow estimation approach that can efficiently produce high-quality results, even when the scene contains very large motions. Our method is local, yet independent of search range, and therefore is fast, thanks to the randomized propagation of self-similarity patterns and correspondence offsets, as well as the hierarchical matching scheme. Evaluations on public benchmarks demonstrate the effectiveness and efficiency of our algorithm. We believe our fast yet effective method will find its place in many practical applications.

References

[1] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[2] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," in *Proc. IJCAI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[3] F. Steinbruecker, T. Pock, and D. Cremers, "Large displacement optical flow computation without warping," in *Proc. ICCV*. IEEE, 2009, pp. 1609–1614.

[4] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "Deepflow: Large displacement optical flow with deep matching," in *Proc. ICCV*. IEEE, 2013, pp. 1385–1392.

[5] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 500–513, March 2011.

[6] L. Xu, J. Jia, and Y. Matsushita, "Motion detail preserving optical flow estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1744–1757, Sept 2012.

[7] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, "Patchmatch: a randomized correspondence algorithm for structural image editing," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 28, no. 3, pp. 24:1–24:11, Jul. 2009.

[8] S. Korman and S. Avidan, "Coherency sensitive hashing," in *Proc. ICCV*. IEEE, 2011, pp. 1607–1614.

[9] K. He and J. Sun, "Computing nearest-neighbor fields via propagation-assisted kd-trees," in *Proc. CVPR*. IEEE, 2012, pp. 111–118.

[10] Z. Chen, H. Jin, Z. Lin, S. Cohen, and Y. Wu, "Large displacement optical flow from nearest neighbor fields," in *Proc. CVPR*. IEEE, 2013.

[11] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *Proc. CVPR*. IEEE, 2011, pp. 3017–3024.

[12] M. Tao, J. Bai, P. Kohli, and S. Paris, "Simpleflow: A non-iterative, sublinear optical flow algorithm," *Comp. Graph. Forum*, vol. 31, no. 2pt1, pp. 345–353, May 2012.

[13] K.-J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, April 2006.

[14] L. Bao, Q. Yang, and H. Jin, "Fast edge-preserving patchmatch for large displacement optical flow," in *Proc. CVPR*. IEEE, June 2014.

[15] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *Int. J. Comput. Vision*, vol. 2, no. 3, pp. 283–310, 1989.

[16] W. Enkelmann, "Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences," in *Comput. Vision Graph. Image Process.*, vol. 43, no. 2. San Diego, CA, USA: Academic Press Professional, Inc., Aug. 1988, pp. 150–177.

[17] L. Alvarez, J. Weickert, and J. Sánchez, "Reliable estimation of dense optical flow fields with large displacements," *Int. J. Comput. Vision*, vol. 39, no. 1, pp. 41–56, 2000.

[18] M. Bleyer, C. Rhemann, and C. Rother, "Patchmatch stereo-stereo matching with slanted support windows." in *Proc. BMVC*, 2011.

[19] J. Lu, H. Yang, D. Min, and M. N. Do, "Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation," in *Proc. CVPR*. IEEE, June 2013, pp. 2443–2450.

[20] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[21] Q. Yang, N. Ahuja, R. Yang, K.-H. Tan, J. Davis, B. Culbertson, J. Apostolopoulos, and G. Wang, "Fusion of median and bilateral filtering for range image upsampling," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 4841–4852, Dec 2013.

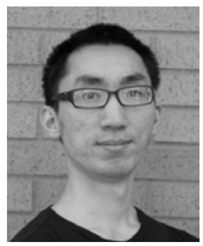
[22] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time weighted median filtering for stereo matching and beyond," in *Proc. ICCV*. IEEE, 2013, pp. 49–56.

[23] Q. Yang, R. Yang, J. Davis, and D. Nistér, "Spatial-depth super resolution for range images," in *Proc. CVPR*. IEEE, June 2007, pp. 1–8.

[24] M. J. Black and P. Anandan, "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields," *Comput. Vis. Image Underst.*, vol. 63, no. 1, pp. 75–104, Jan. 1996.

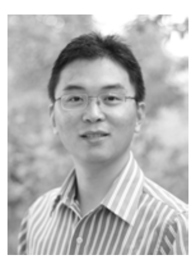
[25] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *Int. J. Comput. Vision*, vol. 106, no. 2, pp. 115–137, Jan. 2014.

[26] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *Proc. ECCV*. Berlin, Heidelberg: Springer-Verlag, 1994, pp. 151–158.
[27] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, “A database and evaluation methodology for optical flow,” *Int. J. Comput. Vision*, vol. 92, no. 1, pp. 1–31, 2011.
[28] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, “Digital photography with flash and no-flash image pairs,” vol. 23, no. 3. New York, NY, USA: ACM, Aug. 2004, pp. 664–672.
[29] K. Zhang, G. Lafruit, R. Lauwereins, and L. Van Gool, “Constant time joint bilateral filtering using joint integral histograms,” *IEEE Trans. Image Process.*, vol. 21, no. 9, pp. 4309–4314, Sept 2012.
[30] S. Paris and F. Durand, “A fast approximation of the bilateral filter using a signal processing approach,” *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 24–52, Jan. 2009.
[31] L. Bao, Y. Song, Q. Yang, and N. Ahuja, “An edge-preserving filtering framework for visibility restoration,” in *Proc. ICPR*. IEEE, Nov 2012, pp. 384–387.
[32] B. Gunturk, “Fast bilateral filter with arbitrary range and domain kernels,” *IEEE Trans. Image Process.*, vol. 20, no. 9, pp. 2690–2696, Sept 2011.
[33] Q. Yang, “Hardware-efficient bilateral filtering for stereo matching,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 5, pp. 1026–1032, May 2014.
[34] M. Leordeanu, A. Zanfir, and C. Sminchisescu, “Locally affine sparse-to-dense matching for motion and occlusion estimation,” in *Proc. ICCV*. IEEE, 2013, pp. 1721–1728.
[35] D. Sun, J. Wulff, E. B. Sudderth, H. Pfister, and M. J. Black, “A fully connected layered model of foreground and background flow,” in *Proc. CVPR*. IEEE, 2013, pp. 2451–2458.
[36] N. Sundaram, T. Brox, and K. Keutzer, “Dense point trajectories by gpu-accelerated large displacement optical flow,” in *Proc. ECCV*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 438–451.
[37] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, “Anisotropic huber-l1 optical flow,” in *Proc. BMVC*, 2009.
[38] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Proc. CVPR*. IEEE, June 2012, pp. 3354–3361.
[39] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, “A naturalistic open source movie for optical flow evaluation,” in *Proc. ECCV*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 611–625.
[40] X. Mei, X. Sun, M. Zhou, H. Wang, X. Zhang *et al.*, “On building an accurate stereo matching system on graphics hardware,” in *Proc. ICCV Workshop*. IEEE, Nov 2011, pp. 467–474.
[41] G. Rong and T.-S. Tan, “Jump flooding in gpu with applications to voronoi diagram and distance transform,” in *Proc. I3D*. New York, NY, USA: ACM, 2006, pp. 109–116.
[42] J. Braux-Zin, R. Dupont, and A. Bartoli, “A general dense image matching framework combining direct and feature-based costs,” in *Proc. ICCV*. IEEE, 2013, pp. 185–192.
[43] O. Demetz, D. Hafner, and J. Weickert, “The complete rank transform: A tool for accurate and morphologically invariant matching of structure,” in *Proc. BMVC*, 2013.
[44] D. Sun, S. Roth, and M. J. Black, “Secrets of optical flow estimation and their principles,” in *Proc. CVPR*. IEEE, June 2010, pp. 2432–2439.
[45] S. Hermann and R. Klette, “Hierarchical scan line dynamic programming for optical flow using semi-global matching,” in *Proc. ACCV*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 556–567.
[46] R. Ranftl, S. Gehrig, T. Pock, and H. Bischof, “Pushing the limits of stereo using variational stereo estimation,” in *Proc. Intelligent Vehicles Symposium (IV)*. IEEE, June 2012, pp. 401–407.
[47] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers, “An improved algorithm for TV-L1 optical flow,” in *Statistical and Geometrical Approaches to Visual Motion Analysis*, ser. Lecture Notes in Computer Science, vol. 5604. Springer Berlin Heidelberg, 2009, pp. 23–45.
[48] T. Senst, J. Geistert, I. Keller, and T. Sikora, “Robust local optical flow estimation using bilinear equations for sparse motion estimation,” in *Proc. ICIP*. IEEE, Sept 2013, pp. 2499–2503.
[49] T. Senst, V. Eiselein, and T. Sikora, “Robust local optical flow for feature tracking,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 9, pp. 1377–1387, Sept 2012.
[50] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, “High accuracy optical flow estimation based on a theory for warping,” in *Proc. ECCV*. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 25–36.
[51] G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Proc. Scandinavian Conference on Image Analysis (SCIA)*. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 363–370.
[52] A. Wedel, D. Cremers, T. Pock, and H. Bischof, “Structure- and motion-adaptive regularization for high accuracy optic flow,” in *Proc. ICCV*. IEEE, Sept 2009, pp. 1663–1668.
[53] P. Gwosdek, H. Zimmer, S. Grewenig, A. Bruhn, and J. Weickert, “A highly efficient gpu implementation for variational optic flow based on the euler-lagrange framework,” in *Trends and Topics in Computer Vision*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 6554, pp. 372–383.
[54] L. Bao, H. Jin, B. Kim, and Q. Yang, “A comparison of tv-l1 optical flow solvers on gpu,” in *Proc. GPU Technology Conference (GTC) Posters*. NVIDIA, 2014, p. P4254.
[55] C. Liu and W. T. Freeman, “A high-quality video denoising algorithm based on reliable motion estimation,” in *Proc. ECCV*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 706–719.



Linchao Bao (S’-14) is currently a Ph.D. student in the Department of Computer Science at City University of Hong Kong. He obtained a M.S. degree in Pattern Recognition and Intelligent Systems from Huazhong University of Science and Technology, Wuhan, China in 2011. His research interests reside in computer vision and graphics.

Qingxiong Yang (M’-11) received the BE degree in Electronic Engineering & Information Science from University of Science & Technology of China in 2004 and the PhD degree in Electrical & Computer Engineering from University of Illinois at Urbana-Champaign in 2010. He is an assistant Professor in the Computer Science Department at City University of Hong Kong. His research interests reside in computer vision and computer graphics. He is a recipient of the best student paper award at MMSP 2010 and the best demo award at CVPR 2007.



Hailin Jin (M’-04) received his Bachelor’s degree in Automation from Tsinghua University, Beijing, China in 1998. He then received his Master of Science and Doctor of Science degrees in Electrical Engineering from Washington University in Saint Louis in 2000 and 2003 respectively. Between fall 2003 and fall 2004, he was a postdoctoral researcher at the Computer Science Department, University of California at Los Angeles. Since October 2004, he has been with Adobe Systems Incorporated where he is currently a Principal Scientist. He received the best student paper award (with J. Andrews and C. Sequin) at the 2012 International CAD conference for work on interactive inverse 3D modeling. He is a member of the IEEE and the IEEE Computer Society.