This CVPR2014 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Fast Edge-Preserving PatchMatch for Large Displacement Optical Flow

Linchao Bao
City University of Hong Kong
linchaobao@gmail.com

Qingxiong Yang*
City University of Hong Kong
qiyang@cityu.edu.hk

Hailin Jin
Adobe Research
hljin@adobe.com

Abstract

We 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.

1. Introduction

Optical flow estimation is one of the most fundamental problems in Computer Vision. Since the seminal work of Horn-Schunck *global* model [12] and Lucas-Kanade *local* model [16], 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 are 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 [21]. 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 deepmatching [25] 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 [7, 27].

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 computationally expensive to compute, there exist efficient approximate algorithms [4, 13, 11].

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 [13] and KD-Tree [11] 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 occlusion 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 [9]. However, doing so may lead to a computationally expensive algorithm which has limited practical applicability. Instead, motivated by recent successes of local methods in

*Corresponding author. This work was supported in part by a GRF grant from the Research Grants Council of Hong Kong under Grant U 122212 and an Adobe gift fund.

stereo matching and optical flow [19, 24, 31] 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 which are motion boundary preservation and algorithm speed. We address the former problem by introducing a novel edge-preserving version of PatchMatch [4] and the latter one by developing a fast approximate algorithm.

1.1. 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 originally formulated in [1]. 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.* [21] 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 [7] 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 [27] incorporated a discrete optimization step before diving into the variational optical flow solver.

Regarding NNFs, PatchMatch [4] 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 [13, 11], which is different from the dense correspondence problem. Exceptionally, the work [6] 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 [9], 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 [31] showed that a good local method can perform equally well. Rhemann *et al.* [19] successfully applied this principle to optical flow and obtained an algorithm that ranks high on the Middlebury flow benchmark. The SimpleFlow [24] followed the same direction but towards a less accurate yet faster solution. The PatchMatch Filter [15] adapted the PatchMatch algorithm onto superpixels and employed the algorithm from [19] to refine the matching correspondence between superpixels.

1.2. 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 [4]. This significantly reduces the computational complexity. We hope this idea to inspire other work in generalizing [4] to other applications.

2. 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 [19], 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 [4]. 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. 1a. To overcome the problems, we propose a new edge-preserving version of PatchMatch (Sec. 2.1) and a corresponding fast algorithm (Sec. 2.2).

The techniques used in [19] for the refinement step (*i.e.*, consistency check and weighted median filtering [29, 17]) are also employed in this paper except that we suggest to

(a) Original PatchMatch (b) Ours (c) Ours (refined)

Figure 1: PatchMatch results (cropped) on the "Army" dataset from Middlebury benchmark [2]. 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.

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 [30]. Details are presented in Sec. 2.3 and 2.4.

2.1. Edge-Preserving PatchMatch

The main idea of original PatchMatch [4] 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

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

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* [31] into the matching cost calculation . Moreover, similar to the data term employed in variational optical flow estimation [5, 22], 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 [5]) to reject outliers. Further more, 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 [32]. Specifically, the matching cost in our approach is defined as follows,

$$d(\mathbf{a},\mathbf{b}) = \frac{1}{W}\sum_{\mathbf{\Delta}} w(\mathbf{a},\mathbf{b},\mathbf{\Delta})C(\mathbf{a},\mathbf{b},\mathbf{\Delta}), \quad (2)$$

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

$$\begin{aligned} w(\mathbf{a},\mathbf{b},\mathbf{\Delta}) &= \exp(-\frac{\|\mathbf{\Delta}\|^2}{\sigma_s^2})\exp(-\frac{\|I^A(\mathbf{a}+\mathbf{\Delta}) - I^A(\mathbf{a})\|^2}{\sigma_r^2}) \\ &\quad \exp(-\frac{\|I^B(\mathbf{b}+\mathbf{\Delta}) - I^B(\mathbf{b})\|^2}{\sigma_r^2}), \quad (3) \end{aligned}$$

$$C(\mathbf{a},\mathbf{b},\mathbf{\Delta}) = \sum_{i=1}^{K}\rho_i(C_i^A(\mathbf{a}+\mathbf{\Delta}) - C_i^B(\mathbf{b}+\mathbf{\Delta})), \quad (4)$$

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. 3).

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 [19]), 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.

2.2. 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.

2.2.1 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 that we 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 from its neighborhood, store their positions and only use the stored pixels to compute the cost.

We performed experiments on the Middlebury training datasets [2] to validate this idea. For each pixel, the neighboring n most similar pixels are used for computing patch matching cost. Table 1 shows the optical flow accuracy and the corresponding runtime on Middlebury training datasets when n is with different value (patch size is fixed to 35×35).

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

Table 1: 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. 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 [19] takes about 430 seconds on the same CPU to produce bidirectional optical flow with search range 61×61.

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 preselection of n similar pixels for each pixel actually can be too slow, especially when patch size is large, which may cancel out a large portion of the speed gain of the PatchMatch. For example, selecting $n = 50$ out of 35×35 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 an 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' vector into its own own vector (according to the stored pixels' similarity to current pixel); reversely scan and merge. Since we do not intend to search exactly the top n similar pixels for each pixel, the algorithm does not need to interleave additional random guesses during propagation or iterate more. Moreover, when merging vectors, if the similarity between two center pixels are very high, we can directly merge the two vectors without re-computing the similarity between surrounding pixels and center pixel. 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). When it comes to the optical flow estimation, we do not experience degraded accuracy on the Middlebury training datasets (in Table 1). More results are provided in Sec. 3.

Algorithm 1 Self-Similarity Propagation Algorithm

Input: image A and B, patch radius r, number of selected similar pixels n.
Output: self-similarity vector S_A and S_B.
/* Initialization */
for each pixel (x, y) in A and B **do**
  (1) randomize a vector $S(x, y)$ containing the location of n neighboring pixels inside the support window;
  (2) sort pixels in $S(x, y)$ according to the Euclidean similarity in CIELab color space to pixel (x, y).
end for
/* Propagation */
for each pixel (x, y) in A and B (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 those pixels' similarity to pixel (x, y);
  (2) for the pixels out of the support window of pixel (x, y), randomize another location inside the window.
end for
for each pixel (x, y) in A and B (scan reversely) **do**
  (1) merge vector $S(x+1, y)$ and $S(x, y+1)$ into $S(x, y)$ according to those pixels' similarity to pixel (x, y);
  (2) for the pixels out of the support window of pixel (x, y), randomize another location inside the window.
end for
/ Result $S(x, y)$ for A is $S_A(x, y)$, and for B is $S_B(x, y)$ */*

2.2.2 Hierarchical Matching

When input image is large, performing PatchMatch on all pixels is a waste of computation resources. 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, typically downsample 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 [30] 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 [24]. 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 (we typically only downsample twice). This will also avoid large error accumulation when propagating NNF estimate from 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. 3).

2.3. Handling Occlusions and Outliers

After computing bidirectional NNFs (at each resolution) between two images, we explicitly perform forward-backward consistency check [19] between the two NNFs to detect occlusion 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 [3] is thus performed on the flow fields to remove the outliers (filtering is performed on *all* pixels). As a final keeper, a second pass consistency check and fixing is performed to make sure the filtering is doing things right. 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.

2.4. Subpixel Refinement

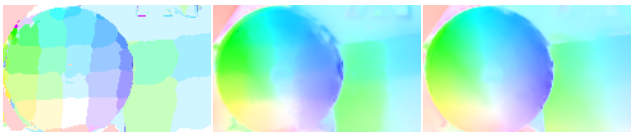
Suppose the discrete correspondence for each pixel $\mathbf{a}$ in image A is $NN_{A\to B}(\mathbf{a}) = \mathbf{b}$, and the patch centered at pixel $\mathbf{a}$ is denoted by Ω_a. We then compute the matching costs between patch Ω_a and m different patches around patch Ω_b, respectively, which is denoted 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, which is

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

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, 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).



(a) Discrete NNF (b) Subpixel refined (c) Improved

Figure 2: Example of subpixel refinement. Note that the improved result in (c) is obtained in the *same* runtime as that in (b).

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 $\mathbf{b}$. Fig. 2 shows the improvement of this strategy.

Finally, an edge-preserving filtering with small parameters (*e.g.*, bilateral filtering [28] 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.

3. Experimental Results

In this section, we present our experimental results on three public optical flow benchmarks – the Middlebury benchmark [2], the KITTI benchmark [10], and the MPI Sintel benchmark [8]. Note that the Middlebury benchmark only contains 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 [18] 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.

3.1. Results on MPI Sintel Benchmark

The MPI Sintel benchmark is a challenging optical flow evaluation benchmark, especially due to the complex elements 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 2 shows the performance of our method on this benchmark (complete table is available online and in supplementary material). 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 2), our method ranks even higher.

Clean pass	EPE all	EPE s40+	Runtime (sec)
DeepFlow [25]	5.377	33.701	17
MDP-Flow2 [27]	5.837	39.459	547
Ours	6.494	39.152	**0.25**
S2D-Matching [14]	6.510	44.187	1920
Classic+NLP [22]	6.731	45.290	888
FC-2Layers-FF [23]	6.781	45.962	4525
LDOF [7]	7.563	51.696	60
Classic+NL [22]	7.961	57.374	888
Classic++ [22]	8.721	60.645	510
Horn+Schunck [12]	8.739	58.243	156
Classic+NL-fast [22]	9.129	66.935	174
SimpleFlow [24]	12.617	81.786	2.9
Aniso. Huber-L1 [26]	12.642	77.835	3.2

Final pass	EPE all	EPE s40+	Runtime (sec)
DeepFlow [25]	7.212	44.118	17
S2D-Matching [14]	7.872	48.782	1920
FC-2Layers-FF [23]	8.137	51.349	4525
Classic+NLP [22]	8.291	51.162	888
Ours	8.377	49.083	**0.25**
MDP-Flow2 [27]	8.445	50.507	547
LDOF [7]	9.116	57.296	60
Classic+NL [22]	9.153	60.291	888
Horn+Schunck [12]	9.610	58.274	156
Classic++ [22]	9.959	64.135	510
Classic+NL-fast [22]	10.088	67.801	174
Aniso. Huber-L1 [26]	11.927	74.796	3.2
SimpleFlow [24]	13.364	81.350	2.9

Table 2: Performance on MPI Sintel benchmark (`http://sintel.is.tue.mpg.de/results`, captured on Oct 30th, 2013). 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 other benchmark website since it is not reported on this benchmark (note that not all of them are reported on GPU).

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. 3), 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. 3b) and heavy fog (Figs. 3d). 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. 4), and this will cause problems at textureless regions for local method since matching cues might be locally dominated by the subtle textures introduced. See Sec. 3.4 for more discussion.

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

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

Figure 3: 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. 4), 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).

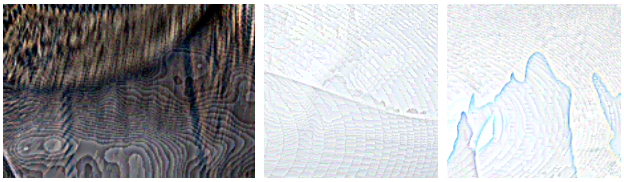


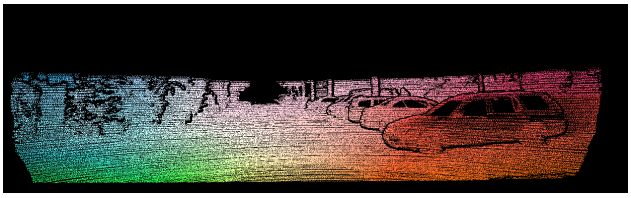
Figure 4: Close-ups (marked by red squares in Fig. 3) for the detail enhanced version of the input images in Fig. 3. The subtle textures introduced by synthetic atmospheric effects can be easily observed.

3.2. 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 flow are caused by camera movement and tend to be smooth with few motion boundaries, which makes traditional coarse-to-fine global method a perfect choice. Besides, the camera distortions near scene boundaries are large, which makes patch matching often fail near such regions. Thus, as a local method based on patch matching, the accuracy performance of our method is not that prominent on this benchmark (see Table 3 for a comparison with other fast methods, complete table is available online and in supplementary material). However, if one is willing to compromise a little on quality for the sake of speed, our

(a) Frame 1

(b) Frame 2



(c) Ground truth

(d) Our result

Figure 5: An example of our result on KITTI benchmark.

	Out-Noc	Avg-Noc	Runtime (sec)
TGV2ADCSIFT	4.71%	1.6px	12
DeepFlow	5.38%	1.5px	17
CRTflow	6.90%	2.7px	18
Classic++	8.04%	2.6px	510
fSGM	8.44%	3.2px	60
Ours	8.62%	2.5px	**0.25**
TGV2CENSUS	9.19%	2.9px	4
Classic+NL-fast	10.13%	3.2px	174
Horn-Schunck	12.47%	4.0px	156
LDOF	18.72%	5.5px	60
TV-L1	26.50%	7.8px	16
BERLOF	30.63%	8.5px	0.231
RLOF	31.49%	8.7px	0.488
HAOF (Brox *et al.*)	32.48%	11.1px	16.2
PolyExpand	44.53%	17.2px	1
Pyramid-LK	57.22%	21.7px	90

Table 3: Performance on KITTI benchmark (`http://www.cvlibs.net/datasets/kitti`, captured on Oct 30th, 2013) when error threshold is 5 pixel. Only pure optical flow algorithms with runtime less than 100 seconds are shown (i.e., methods incorporated with stereo matching or epipolar geometry are not shown). Note that the runtime are reproduced from the benchmark website (not all of them are reported on GPU).

method can provide a good choice in this case. Fig. 5 gives an example of our results.

3.3. 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 4 shows the performance of our method on the Middlebury benchmark (complete table is available online and in supplementary material). 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 4, notice that hierarchical matching scheme is for fast approximation).

Method	Avg. Rank	Avg. EPE	Reported Time (sec)
Ours (w/o HM)	31.1	0.33	2.5
SimpleFlow	35.5	0.47	1.7+240‡
Adaptive	38.9	0.40	9.2
CompOF-FED-GPU	43.0	0.47	0.97
Aniso. Huber-L1	44.2	0.40	2
TV-L1-Improved	49.1	0.54	2.9
Ours	64.0	0.62	**0.20**

‡: The reported results of SimpleFlow are obtained after global optimization using [22], which takes about 240 seconds using the code provided by [22].

Table 4: Performance on Middlebury benchmark (`http://vision.middlebury.edu/flow`, Endpoint Error, captured on Oct 30th, 2013). Only algorithms with similar reported runtime to our method are shown. Note that the runtime are reproduced from the benchmark website (all of them are reported on GPU).

(a) Input (first frame)

(b) Ground truth

(c) with HM (0.82)

(d) without HM (0.63)

Figure 6: An example of our results on Middlebury benchmark. The EPE is shown in the caption.

3.4. Limitations

As a local method, our approach will fail at large textureless regions, where local evidences are not enough to eliminate matching ambiguities. While increasing patch size and 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. 4), which may lead to incorrect match. In this case, global optimization techniques may help to solve the problem. However, notice that mismatch in large textureless regions might not be a serious problem for some real-world applications.

4. 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] L. Alvarez, J. Weickert, and J. Sánchez. Reliable estimation of dense optical flow fields with large displacements. *IJCV*, 2000.

[2] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 2011.

[3] L. Bao, Y. Song, Q. Yang, and N. Ahuja. An edge-preserving filtering framework for visibility restoration. In *ICPR*, 2012.

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM TOG*, 2009.

[5] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *CVIU*, 1996.

[6] M. Bleyer, C. Rhemann, and C. Rother. Patchmatch stereo-stereo matching with slanted support windows. In *BMVC*, 2011.

[7] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *TPAMI*, 2011.

[8] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012.

[9] Z. Chen, H. Jin, Z. Lin, S. Cohen, and Y. Wu. Large displacement optical flow from nearest neighbor fields. In *CVPR*, 2013.

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[11] K. He and J. Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *CVPR*, 2012.

[12] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 16:185–203, 1981.

[13] S. Korman and S. Avidan. Coherency sensitive hashing. In *ICCV*, 2011.

[14] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Locally affine sparse-to-dense matching for motion and occlusion estimation. In *ICCV*, 2013.

[15] 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 *CVPR*, 2013.

[16] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[17] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu. Constant time weighted median filtering for stereo matching and beyond. In *ICCV*, 2013.

[18] X. Mei, X. Sun, M. Zhou, H. Wang, X. Zhang, et al. On building an accurate stereo matching system on graphics hardware. In *ICCV Workshop*, 2011.

[19] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, 2011.

[20] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

[21] F. Steinbruecker, T. Pock, and D. Cremers. Large displacement optical flow computation without warping. In *ICCV*, 2009.

[22] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *IJCV*, 2013.

[23] D. Sun, J. Wulff, E. B. Sudderth, H. Pfister, and M. J. Black. A fully connected layered model of foreground and background flow. In *CVPR*, 2013.

[24] M. Tao, J. Bai, P. Kohli, and S. Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. In *Computer Graphics Forum*, 2012.

[25] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *ICCV*, 2013.

[26] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic huber-l1 optical flow. In *BMVC*, 2009.

[27] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *TPAMI*, 2012.

[28] Q. Yang. Hardware-efficient bilateral filtering for stereo matching. *TPAMI*, 2014.

[29] 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. *TIP*, 2013.

[30] Q. Yang, R. Yang, J. Davis, and D. Nistér. Spatial-depth super resolution for range images. In *CVPR*, 2007.

[31] K.-J. Yoon and I. S. Kweon. Adaptive support-weight approach for correspondence search. *TPAMI*, 2006.

[32] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *ECCV*, 1994.