EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow

Jerome Revaud[a] Philippe Weinzaepfel[a] Zaid Harchaoui[a,b] Cordelia Schmid[a]
[a] Inria* [b] NYU
firstname.lastname@inria.fr

Abstract

We propose a novel approach for optical flow estimation, targeted at large displacements with significant occlusions. It consists of two steps: i) dense matching by edge-preserving interpolation from a sparse set of matches; ii) variational energy minimization initialized with the dense matches. The sparse-to-dense interpolation relies on an appropriate choice of the distance, namely an edge-aware geodesic distance. This distance is tailored to handle occlusions and motion boundaries – two common and difficult issues for optical flow computation. We also propose an approximation scheme for the geodesic distance to allow fast computation without loss of performance. Subsequent to the dense interpolation step, standard one-level variational energy minimization is carried out on the dense matches to obtain the final flow estimation. The proposed approach, called Edge-Preserving Interpolation of Correspondences (EpicFlow) *is fast and robust to large displacements. It significantly outperforms the state of the art on MPI-Sintel and performs on par on Kitti and Middlebury.*

1. Introduction

Accurate estimation of optical flow from real-world videos remains a challenging problem [10], despite the abundant literature on the topic. The main remaining challenges are occlusions, motion discontinuities and large displacements, all present in real-world videos.

Effective approaches were previously proposed for handling the case of small displacements (*i.e.*, less than a few pixels) [19, 35, 24]. These approaches cast the optical flow problem into an energy minimization framework, often solved using efficient coarse-to-fine algorithms [8, 28]. However, due to the complexity of the minimization, such methods get stuck in local minima and may fail to estimate large displacements, which often occur due to fast motion. This problem has recently received significant attention. State-of-the-art approaches [9, 38] use descriptor matching between adjacent frames together with the integration of these matches in a variational approach. Indeed, matching operators are robust to large displacements and motion discontinuities [9, 34]. Energy minimization is carried out in a coarse-to-fine scheme in order to obtain a full-scale dense flow field guided by the matches. A major drawback of coarse-to-fine schemes is error-propagation, *i.e.*, errors at coarser levels, where different motion layers can overlap, can propagate across scales. Even if coarse-to-fine techniques work well in most cases, we are not aware of a theoretical guarantee or proof of convergence.

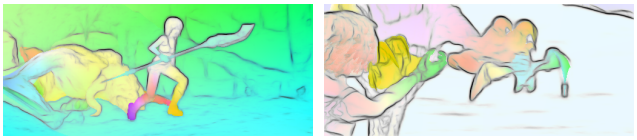

Figure 1. Image edges detected with SED [15] and ground-truth optical flow. Motion discontinuities appear most of the time at image edges.

Instead, we propose to simply interpolate a sparse set of matches in a dense manner to initiate the optical flow estimation. We then use this estimate to initialize a one-level energy minimization, and obtain the final optical flow estimation. This enables us to leverage recent advances in matching algorithms, which can now output quasi-dense correspondence fields [6, 34]. In the same spirit as [22], we perform a sparse-to-dense interpolation by fitting a local affine model at each pixel based on nearby matches. A major issue arises for the preservation of motion boundaries. We make the following observation: *motion boundaries often tend to appear at image edges*, see Figure 1. Consequently, we propose to exchange the Euclidean distance with a better, *i.e.*, edge-aware, distance and show that this offers a natural way to handle motion discontinuities. Moreover, we show how an approximation of the edge-aware distance allows to fit only one affine model per input match (instead of one per pixel). This leads to an important speed-up of the interpolation scheme without loss in performance.

The obtained interpolated field of correspondences is sufficiently accurate to be used as initialization of a one-level energy minimization. Our work suggests that there may be better initialization strategies than the well-

*LEAR team, Inria Grenoble Rhone-Alpes, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes, France.

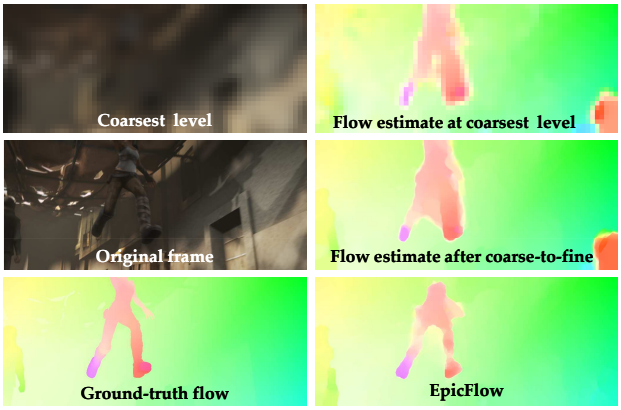

Figure 2. Comparison of coarse-to-fine flow estimation and EpicFlow. Errors at the coarsest level of estimation, due to a low resolution, often get propagated to the finest level (right, top and middle). In contrast, our interpolation scheme benefits from an edge prior at the finest level (right, bottom).

established coarse-to-fine scheme, see Figure 2. In particular, our approach, *EpicFlow* (edge-preserving interpolation of correspondences) performs best on the challenging MPI-Sintel dataset [10] and is competitive on Kitti [16] and Middlebury [4]. An overview of EpicFlow is given in Figure 3. To summarize, we make three main contributions:

- We propose *EpicFlow*, a novel sparse-to-dense interpolation scheme of matches based on an edge-aware distance. We show that it is robust to motion boundaries, occlusions and large displacements.
- We propose an approximation scheme for the edge-aware distance, leading to a significant speed-up without loss of accuracy.
- We show empirically that the proposed optical flow estimation scheme is more accurate than estimations based on coarse-to-fine minimization.

This paper is organized as follows. In Section 2, we review related work on large displacement optical flow. We then present the sparse-to-dense interpolation in Section 3 and the energy minimization for optical flow computation in Section 4. Finally, Section 5 presents experimental results. Source code is available online at `http://lear.inrialpes.fr/software`.

2. Related Work

Most optical flow approaches are based on a variational formulation and a related energy minimization problem [19, 4, 28]. The minimization is carried out using a coarse-to-fine scheme [8]. While such schemes are attractive from a computational point of view, the minimization often gets stuck in local minima and leads to error accumulation across scales, especially in the case of large displacements [2, 9].

To tackle this issue, the addition of descriptor/matching was recently investigated in several papers. A penalization

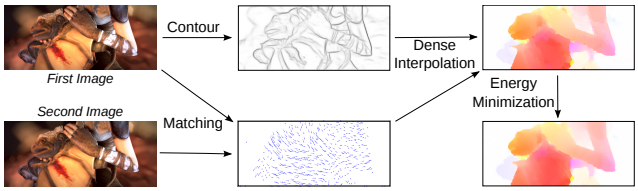

Figure 3. Overview of EpicFlow. Given two images, we compute matches using DeepMatching [34] and the edges of the first image using SED [15]. We combine these two cues to densely interpolate matches and obtain a dense correspondence field. This is used as initialization of a one-level energy minimization framework.

of the difference between flow and HOG matches was added to the energy by Brox and Malik [9]. Weinzaepfel *et al.* [34] replaced the HOG matches by an approach based on similarities of non-rigid patches: DeepMatching. Xu *et al.* [38] merged the estimated flow with matching candidates at each level of the coarse-to-fine scheme. Braux-Zin *et al.* [7] used segment features in addition to keypoints. However, these methods rely on a coarse-to-fine scheme, that suffers from intrinsic flaws. Namely, details are lost at coarse scales, and thin objects with substantially different motions cannot be detected. Those errors correspond to local minima, hence they cannot be recovered and are propagated across levels, see Figure 2.

In contrast, our approach is conceptually closer to recent work that rely mainly on descriptor matching [36, 23, 12, 22, 37, 27, 5]. Lu *et al.* [23] propose a variant of PatchMatch [6], which uses SLIC superpixels [1] as basic blocks in order to better respect image boundaries. The purpose is to produce a nearest-neighbor-field (NNF) which is later translated into a flow. However, SLIC superpixels are only *locally* aware of image edges, whereas our edge-aware distance is able to capture regions at the image scale. Similarly, Chen *et al.* [12] propose to compute an approximate NNF, and then estimate the dominant motion patterns using RANSAC. They, then, use a multi-label graph-cut to solve the assignment of each pixel to a motion pattern candidate. Their multi-label optimization can be interpreted as a motion segmentation problem or as a layered model [29]. These problems are hard and a small error in the assignment can lead to large errors in the resulting flow.

In the same spirit as our approach, Ren [26] proposes to use edge-based affinities to group pixels and estimate a piece-wise affine flow. Nevertheless, this work relies on a discretization of the optical flow constraint, which is valid only for small displacements. Closely related to EpicFlow, Leordeanu *et al.* [22] also investigate sparse-to-dense interpolation. Their initial matching is obtained through the costly minimization of a global non-convex matching energy. In contrast, we directly use state-of-the-art matches [34, 18] as input. Furthermore, during their sparse-to-dense interpolation, they compute an affine trans-

formation independently for each pixel based on its neighborhood matches, which are found in a Euclidean ball and weighted by an estimation of occluded areas that involves learning a binary classifier. In contrast, we propose to use an edge-preserving distance that naturally handles occlusions, and can be very efficiently computed.

3. Sparse-to-dense interpolation

3.1. Interpolation method

We propose to estimate a dense correspondence field $\boldsymbol{F} : I \rightarrow I'$ between two images I and I' by interpolating a given set of inputs matches $\mathcal{M} = \{(\boldsymbol{p}_m, \boldsymbol{p}'_m)\}$. Each match $(\boldsymbol{p}_m, \boldsymbol{p}'_m) \in \mathcal{M}$ defines a correspondence between a pixel $\boldsymbol{p}_m \in I$ and a pixel $\boldsymbol{p}'_m \in I'$. The interpolation requires a distance $D : I \times I \rightarrow \mathbb{R}^+$ between pixels, see Section 3.2. We consider here two options for the interpolation.

• **Nadaraya-Watson (NW) estimation [31].** The correspondence field $\boldsymbol{F}_{NW}(\boldsymbol{p})$ is interpolated using the Nadaraya-Watson estimator at a pixel $\boldsymbol{p} \in I$ and is expressed by a sum of matches weighted by their proximity to $\boldsymbol{p}$:

$$\boldsymbol{F}_{NW}(\boldsymbol{p}) = \frac{\sum\limits_{(\boldsymbol{p}_m, \boldsymbol{p}'_m) \in \mathcal{M}} k_D(\boldsymbol{p}_m, \boldsymbol{p}) \boldsymbol{p}'_m}{\sum\limits_{(\boldsymbol{p}_m, \boldsymbol{p}'_m) \in \mathcal{M}} k_D(\boldsymbol{p}_m, \boldsymbol{p})} \ , \tag{1}$$

where $k_D(\boldsymbol{p}_m, \boldsymbol{p}) = \exp\left(-aD(\boldsymbol{p}_m, \boldsymbol{p})\right)$ is a Gaussian kernel for a distance D with a parameter a.

• **Locally-weighted affine (LA) estimation [17].** The second estimator is based on fitting a local affine transformation. The correspondence field $\boldsymbol{F}_{LA}(\boldsymbol{p})$ is interpolated using a locally-weighted affine estimator at a pixel $\boldsymbol{p} \in I$ as $\boldsymbol{F}_{LA}(\boldsymbol{p}) = A_{\boldsymbol{p}}\boldsymbol{p} + t_{\boldsymbol{p}}^\top$, where $A_{\boldsymbol{p}}$ and $t_{\boldsymbol{p}}$ are the parameters of an affine transformation estimated for pixel $\boldsymbol{p}$. These parameters are computed as the least-square solution of an overdetermined system obtained by writing two equations for each match $(\boldsymbol{p}_m, \boldsymbol{p}'_m) \in \mathcal{M}$ weighted as previously:

$$k_D(\boldsymbol{p}_m, \boldsymbol{p}) \left(A_{\boldsymbol{p}}\boldsymbol{p}_m + t_{\boldsymbol{p}}^\top - \boldsymbol{p}'_m\right) = 0 \ . \tag{2}$$

Local interpolation. Note that the influence of remote matches is either negligible, or could harm the interpolation, for example when objects move differently. Therefore, we restrict the set of matches used in the interpolation at a pixel $\boldsymbol{p}$ to its K nearest neighbors according to the distance D, which we denote as $\mathcal{N}_K(\boldsymbol{p})$. In other words, we replace the summation over $\mathcal{M}$ in the NW operator by a summation over $\mathcal{N}_K(\boldsymbol{p})$, and likewise for building the overdetermined system to fit the affine transformation for $\boldsymbol{F}_{LA}$.

3.2. Edge-preserving distance

Using the Euclidean distance for the interpolation presented above is possible. However, in this case the interpolation is simply based on the position of the input matches and does not respect motion boundaries. Suppose for a moment that the motion boundaries are known. We can, then, use a geodesic distance D_G based on these motion boundaries. The geodesic distance between two pixels $\boldsymbol{p}$ and $\boldsymbol{q}$ is defined as the shortest distance with respect to a cost map C:

$$D_G(\boldsymbol{p}, \boldsymbol{q}) = \inf_{\Gamma \in \mathcal{P}_{\boldsymbol{p},\boldsymbol{q}}} \int_\Gamma C(\boldsymbol{p}_s) d\boldsymbol{p}_s \ , \tag{3}$$

where $\mathcal{P}_{\boldsymbol{p},\boldsymbol{q}}$ denotes the set of all possible paths between $\boldsymbol{p}$ and $\boldsymbol{q}$, and $C(\boldsymbol{p}_s)$ the cost of crossing pixel $\boldsymbol{p}_s$ (the viscosity in physics). In our settings, C corresponds to the motion boundaries. Hence, a pixel belonging to a motion layer is close to all other pixels from the same layer according to D_G, but far from everything beyond the boundaries. Since each pixel is interpolated based on its neighbors, the interpolation will respect the motion boundaries.

In practice, we use an alternative to true motion boundaries, making the plausible assumption that *image edges* are a superset of *motion boundaries*. This way, the distance between pixels belonging to the same region will be low. It ensures a proper edge-respecting interpolation as long as the number of matches in each region is sufficient. Similarly, Criminisi *et al.* [13] showed that geodesic distances are a natural tool for edge-preserving image editing operations (denoising, texture flattening, etc.) and it was also used recently to generate object proposals [21]. In practice, we set the cost map C using a recent state-of-the-art edge detector, namely the "structured edge detector" (SED) [15][1]. Figure 4 shows an example of a SED map, as well as examples of geodesic distances and neighbor sets $\mathcal{N}_K(\boldsymbol{p})$ for different pixels $\boldsymbol{p}$. Notice how neighbors are found on the same objects/parts of the image with D_G, in contrast to Euclidean distance (see also Figure 6).

3.3. Fast approximation

The geodesic distance can be rapidly computed from a point to all other pixels. For instance, Weber *et al.* [32] propose parallel algorithms that simulate an advancing wavefront. Nevertheless, the computational cost for computing the geodesic distance between all pixels and all matches (as required by our interpolation scheme) is high. We now propose an efficient approximation $\tilde{D}_G$.

A key observation is that neighboring pixels are often interpolated similarly, suggesting a strategy that would leverage such local information. In this section we employ the term "match" to refer to $\boldsymbol{p}_m$ instead of $(\boldsymbol{p}_m, \boldsymbol{p}'_m)$.

Geodesic Voronoi diagram. We first define a clustering L, such that $L(\boldsymbol{p})$ assigns a pixel $\boldsymbol{p}$ to its closest match according to the geodesic distance, *i.e.*, we have $L(\boldsymbol{p}) = \operatorname{argmin}_{\boldsymbol{p}_m} D_G(\boldsymbol{p}, \boldsymbol{p}_m)$. L defines geodesic Voronoi cells, as shown in Figure 5(c).

[1] https://github.com/pdollar/edges

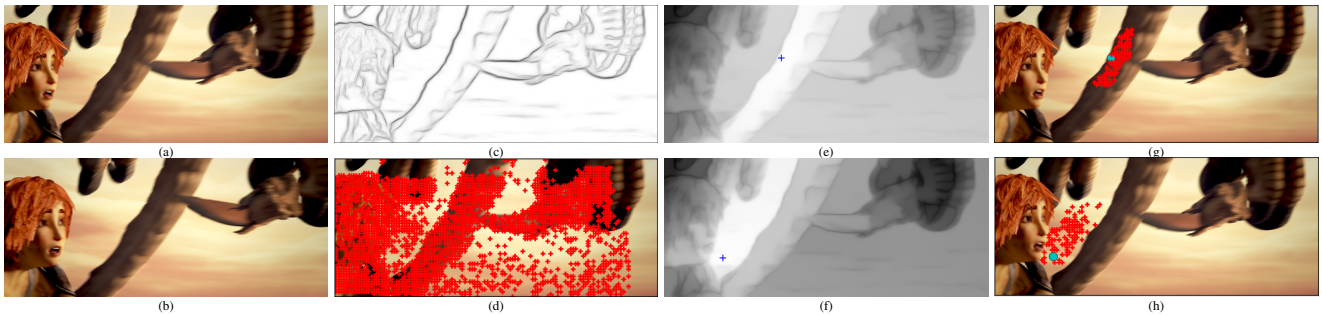

Figure 4. (a-b) two consecutive frames; (c) contour response C from SED [15] (the darker, the higher); (d) match positions $\{\boldsymbol{p}_m\}$ from DeepMatching [34]; (e-f) geodesic distance from a pixel $\boldsymbol{p}$ (marked in blue) to all others $D_G(\boldsymbol{p}, .)$ (the brighter, the closer). (g-h) 100 nearest matches, *i.e.*, $\mathcal{N}_{100}(\boldsymbol{p})$ (red) using geodesic distance D_G from the pixel $\boldsymbol{p}$ in blue.

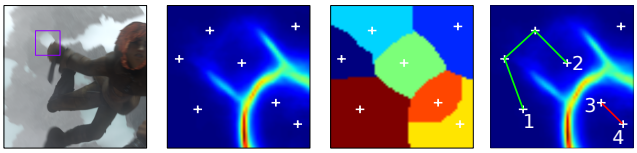

Figure 5. For the region shown in (a), (b) shows the image edges C and white crosses representing the match positions $\{\boldsymbol{p}_m\}$. (c) displays the assignment L, *i.e.*, geodesic Voronoi cells. We build a graph $\mathcal{G}$ from L (see text). (d) shows the shortest path between two neighbor matches, which can go through the edge that connects them (3-4) or a shorter path found by Dijkstra's algorithm (1-2).

Approximated geodesic distance. We then approximate the distance between a pixel $\boldsymbol{p}$ and any match $\boldsymbol{p}_m$ as the distance to the closest match $L(\boldsymbol{p})$ plus an approximate distance between matches:

$$\tilde{D}_G(\boldsymbol{p}, \boldsymbol{p}_m) = D_G(\boldsymbol{p}, L(\boldsymbol{p})) + D_G^{\mathcal{G}}(L(\boldsymbol{p}), \boldsymbol{p}_m) \quad (4)$$

where $D_G^{\mathcal{G}}$ is a graph-based approximation of the geodesic distance between two matches. To define $D_G^{\mathcal{G}}$ we use a neighborhood graph $\mathcal{G}$ whose nodes are $\{\boldsymbol{p}_m\}$. Two matches $\boldsymbol{p}_m$ and $\boldsymbol{p}_n$ are connected by an edge if they are neighbors in L. The edge weight is then defined as the geodesic distance between $\boldsymbol{p}_m$ and $\boldsymbol{p}_n$, where the geodesic distance calculation is restricted to the Voronoi cells of $\boldsymbol{p}_m$ and $\boldsymbol{p}_n$. We, then, calculate the approximate geodesic distance between any two matches $\boldsymbol{p}_m, \boldsymbol{p}_n$ using Dijkstra's algorithm on $\mathcal{G}$, see Figure 5(d).

Piecewise field. So far, we have built an approximation of the distance between pixels and match points. We now show that our interpolation model results in a piece-wise correspondence field (either constant for the Nadaraya-Watson estimator, or piece-wise affine for LA). This property is crucial to obtain a fast interpolation scheme, and experiments shows that it does not impact the accuracy. Let us consider a pixel $\boldsymbol{p}$ such that $L(\boldsymbol{p}) = \boldsymbol{p}_m$. The distance between $\boldsymbol{p}$ and any match $\boldsymbol{p}_n$ is the same as the one between $\boldsymbol{p}_m$ and $\boldsymbol{p}_n$ up to a constant independent from $\boldsymbol{p}_n$ (Equation 4). As a consequence, we have $\mathcal{N}_K(\boldsymbol{p}) = \mathcal{N}_K(\boldsymbol{p}_m)$ and $k_{\tilde{D}_G}(\boldsymbol{p}, \boldsymbol{p}_n) = k_{D_G}(\boldsymbol{p}, \boldsymbol{p}_m) \times k_{D_G^{\mathcal{G}}}(\boldsymbol{p}_m, \boldsymbol{p}_n)$. For the Nadaraya-Watson estimator, we thus obtain:

$$\boldsymbol{F}_{NW}(\boldsymbol{p}) = \frac{\sum_{(\boldsymbol{p}_n, \boldsymbol{p}'_n)} k_{\tilde{D}_G}(\boldsymbol{p}, \boldsymbol{p}_n) \boldsymbol{p}'_n}{\sum_{(\boldsymbol{p}_n, \boldsymbol{p}'_n)} k_{\tilde{D}_G}(\boldsymbol{p}, \boldsymbol{p}_n)} \quad (5)$$

$$= \frac{k_{D_G}(\boldsymbol{p}, \boldsymbol{p}_m) \sum_{(\boldsymbol{p}_n, \boldsymbol{p}'_n)} k_{D_G^{\mathcal{G}}}(\boldsymbol{p}_m, \boldsymbol{p}_n) \boldsymbol{p}'_n}{k_{D_G}(\boldsymbol{p}, \boldsymbol{p}_m) \sum_{(\boldsymbol{p}_n, \boldsymbol{p}'_n)} k_{D_G^{\mathcal{G}}}(\boldsymbol{p}_m, \boldsymbol{p}_n)} = \boldsymbol{F}_{NW}(\boldsymbol{p}_m)$$

where all the sums are for $(\boldsymbol{p}_n, \boldsymbol{p}'_n) \in \mathcal{N}_K(\boldsymbol{p}) = \mathcal{N}_K(\boldsymbol{p}_m)$. The same reasoning holds for the weighted affine interpolator, which is invariant to a multiplication of the weights by a constant factor. As a consequence, it suffices to compute $|\mathcal{M}|$ estimations (one per match) and to propagate it to the pixel assigned to this match. This is orders of magnitude faster than an independent estimation for each pixel, *e.g.* as done in [22]. We summarize the approach in Algorithm 1 for Nadaraya-Watson estimator. The algorithm is similar for LA interpolator (*e.g.* line 6 becomes "Estimate affine parameters $A_{\boldsymbol{p}_m}, t_{\boldsymbol{p}_m}$" and line 8 "Set $\boldsymbol{W}_{LA}(\boldsymbol{p}) = A_{L(\boldsymbol{p})}\boldsymbol{p} + t_{L(\boldsymbol{p})}^{\top}$").

Algorithm 1 Interpolation with Nadaraya-Watson

Input: a pair of images I, I', a set $\mathcal{M}$ of matches
Output: dense correspondence field $\boldsymbol{F}_{NW}$
1 Compute the cost C for I using SED [15]
2 Compute the assignment map L
3 Build the graph $\mathcal{G}$ from L
4 **For** $(\boldsymbol{p}_m, \boldsymbol{p}'_m) \in \mathcal{M}$
5 Compute $\mathcal{N}_K(\boldsymbol{p}_m)$ from $\mathcal{G}$ using Dijkstra's algorithm
6 Compute $\boldsymbol{F}_{NW}(\boldsymbol{p}_m)$ from $\mathcal{N}_K(\boldsymbol{p}_m)$ using Eq. 1
7 **For** each pixel $\boldsymbol{p}$
8 Set $\boldsymbol{F}_{NW}(\boldsymbol{p}) = \boldsymbol{F}_{NW}(L(\boldsymbol{p}))$

4. Optical Flow Estimation

Coarse-to-fine vs. EpicFlow. The output of the sparse-to-dense interpolation is a dense correspondence field. This field is used as initialization of a variational energy minimization method. In contrast to our approach, state-of-the-art methods usually rely on a coarse-to-fine scheme to

compute the full-scale correspondence field. To the best of our knowledge, there exists no theoretical proof or guarantee that a coarse-to-fine minimization leads to a consistent estimation that accurately minimizes the full-scale energy. Thus, the coarse-to-fine scheme should be considered as a heuristic to provide an initialization for the full-scale flow.

Our approach can be thought of as an alternative to the above strategy, by offering a smart heuristic to accurately initialize the optical flow before performing energy minimization at the full-scale. This offers several advantages over the coarse-to-fine scheme. First, the cost map C in our method acts as a prior on boundary location. Such a prior could also be incorporated by a local smoothness weight in the coarse-to-fine minimization, but would then be difficult to interpret at coarse scales where boundaries might strongly overlap. In addition, since our method directly works at the full image resolution, it avoids possible issues related to the presence of thin objects that could be oversmoothed at coarse scales. Such errors at coarse scales are propagated to finer scales as the coarse-to-fine approach proceeds, see Figure 2.

Variational Energy Minimization. We minimize an energy defined as a sum of a data term and a smoothness term. We use the same data term as [40], based on a classical color-constancy and gradient-constancy assumption with a normalization factor. For the smoothness term, we penalize the flow gradient norm, with a local smoothness weight α as in [33, 38]: $\alpha(\boldsymbol{x}) = \exp\big(-\kappa\|\nabla_2 I(\boldsymbol{x})\|\big)$ with $\kappa = 5$. We have also experimented using SED instead and obtained similar performance.

For minimization, we initialize the solution with the output of our sparse-to-dense interpolation and use the approach of [8] without the coarse-to-fine scheme. More precisely, we perform 5 fixed point iterations, *i.e.*, compute the non-linear weights (that appear when applying Euler-Lagrange equations [8]) and the flow updates 5 times iteratively. The flow updates are computed by solving linear systems using 30 iterations of the successive over relaxation method [39].

5. Experiments

In this section, we evaluate EpicFlow on three state-of-the-art datasets:

• *MPI-Sintel dataset* [10] is a challenging evaluation benchmark obtained from an animated movie. It contains multiple sequences including large/rapid motions. We only use the "final" version that features realistic rendering effects such as motion, defocus blur and atmospheric effects.

• The *Kitti dataset* [16] contains photos shot in city streets from a driving platform. It features large displacements, different materials (complex 3D objects like trees), a large variety of lighting conditions and non-lambertian surfaces.

• The *Middlebury dataset* [4] has been extensively used for evaluating optical flow methods. It contains complex motions, but displacements are limited to a few pixels.

As in [34], we optimize the parameters on a subset (20%) of the MPI-Sintel training set. We then report average endpoint error (AEE) on the remaining MPI-Sintel training set (80%), the Kitti training set and the Middlebury training set. This allows us to evaluate the impact of parameters on different datasets and avoid overfitting. The parameters are typically $a \simeq 1$ for the coefficient in the kernel k_D, the number of neighbors is $K \simeq 25$ for NW interpolation and $K \simeq 100$ when using LA. In Section 5.4, we compare to the state of the art on the test sets. In this case, the parameters are optimized on the training set of the corresponding dataset. Timing is reported for one CPU-core at 3.6GHz.

In the following, we first describe two types of input matches in Section 5.1. Section 5.2 then studies the different parameters of our approach. In Section 5.3, we compare our method to a variational approach with a coarse-to-fine scheme. Finally, we show that EpicFlow outperforms current methods on challenging datasets in Section 5.4.

5.1. Input matches

To generate input matches, we use and compare two recent matching algorithms. They each produce about 5000 matches per image.

• The first one is DeepMatching (**DM**), used in DeepFlow [34], which has shown excellent performance for optical flow. It builds correspondences by computing similarities of non-rigid patches, allowing for some deformations. We use the online code[2] on images downscaled by a factor 2. A reciprocal verification is included in DM. As a consequence, the majority of matches in occluded areas are pruned, see matches in Figure 6 (left).

• The second one is a recent variant of PatchMatch [6] that relies on kd-trees and local propagation to compute a dense correspondence field [18] (**KPM**). We use the online code to extract the dense correspondence field[3]. It is noisy, as it is based on small patches without global regularization, as well as often incorrect in case of occlusion. Thus, we perform a two-way matching and eliminate non-reciprocal matches to remove incorrect correspondences. We also subsample these pruned correspondences to speed-up the interpolation. We have experimentally verified on several image pairs that this does not result in a loss of performance.

Pruning of matches. In both cases, matches are extracted locally and might be incorrect in regions with low texture. Thus, we remove matches corresponding to patches with low saliency, which are determined by the eigenvalues of autocorrelation matrix. Furthermore, we perform a consistency check to remove outliers. We run the sparse-to-dense interpolation once with the Nadaraya-Watson estimator and

[2] http://lear.inrialpes.fr/src/deepmatching/

[3] http://j0sh.github.io/thesis/kdtree/

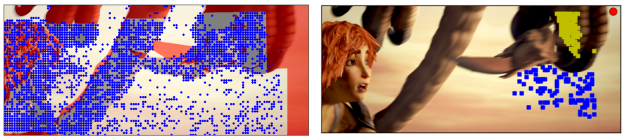

Figure 6. *Left:* Match positions returned by [34] are shown in blue. Red denotes occluded areas. *Right:* Yellow (resp. blue) squares correspond to the 100 nearest matches with a Euclidean (resp. edge-aware geodesic) distance for the occluded pixel shown in red.

	Matching	Interpolator	MPI-Sintel	Kitti	Middlebury
Interpolation	KPM	NW	6.052	15.679	0.765
	KPM	LA	6.334	12.011	0.776
	DM	NW	4.143	5.460	0.898
	DM	LA	4.068	3.560	0.840
EpicFlow	KPM	NW	5.741	15.240	0.388
	KPM	LA	5.764	11.307	**0.315**
	DM	NW	3.804	4.900	0.485
	DM	LA	**3.686**	**3.334**	0.380

Table 1. Comparison of average endpoint error (AEE) for different sparse matches (DM, KPM) and interpolators (NW, LA) as well as for sparse-to-dense interpolation (top) and EpicFlow (bottom). The approximated geodesic distance $\tilde{D}_G$ is used.

remove matches for which the difference to the initial estimate is over 5 pixels.

We also experiment with synthetic sparse matches of various densities and noise levels in Section 5.3, in order to evaluate the sensitivity of EpicFlow to the quality of the matching approach.

5.2. Impact of the different parameters

In this section, we evaluate the impact of the matches and the interpolator. We also compare the quality of the sparse-to-dense interpolation and EpicFlow. Furthermore, we examime the impact of the geodesic distance, of its approximation and of the quality of the contour detector.

Matches and interpolators. Table 1 compares the result of our sparse-to-dense interpolation, *i.e.*, before energy minimization, and EpicFlow for different matches (**DM** and **KPM**) and for the two interpolation schemes: Nadaraya-Watson (**NW**) and locally-weighted affine (**LA**). The approximated geodesic distance is used in the interpolation, see Section 3.3.

We can observe that KPM is consistently outperformed by DeepMatching (DM) on MPI-Sintel and Kitti datasets, with a gap of 2 and 8 pixels respectively. Kitti contains many repetitive textures like trees or roads, which are often mismatched by KPM. Note that DM is significantly more robust to repetitive textures than KPM, as it uses a multi-scale scoring scheme. The results on Middlebury are comparable and below 1 pixel.

We also observe that LA performs better than NW on Kitti, while the results are comparable on MPI-Sintel and Middlebury. This is due to the specificity of the Kitti dataset, where the scene consists of planar surfaces and, thus, affine transformations are more suitable than translations to approximate the flow. Based on these results, we use DM matches and LA interpolation in the remainder of the experimental section.

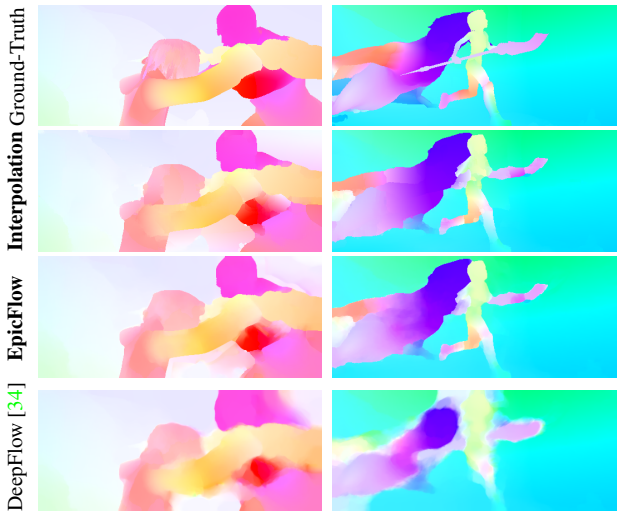

Figure 7. *From top to bottom:* ground-truth flow, result of sparse-to-dense interpolation (Interpolation), full method (EpicFlow), and DeepFlow [34], one of the top performers on MPI-Sintel.

The interpolation is robust to the neighborhood size K with for instance an AEE of $4.082, 4.053, 4.068$ and 4.076 for $K = 50, 100, 160$ (optimal value on the training set), 200 respectively, on MPI-Sintel with the LA estimator and before variational minimization. We also implemented a variant where we use all matches closer than a threshold and obtained similar performance.

Sparse-to-dense interpolation versus EpicFlow. We also evaluate the gain due to the variational minimization using the interpolation as initialization. We can see in Table 1 that this step clearly improves the performance in all cases. The improvement is around 0.5 pixel. Figure 7 presents results for two image pairs with the initialization only and the final result of EpicFlow (second and third rows). While the flow images look similar overall, the minimization allows to further smooth and refine the flow, explaining the gain in performance. Yet, it preserves discontinuities and small details, such as the legs in the right column. In the following, results are reported for EpicFlow, *i.e.*, after the variational minimization step.

Edge-aware versus Euclidean distances. We now study the impact of different distances. First, we examine the effect of approximating the geodesic distance (Section 3.3). Table 2 shows that our approximation has a negligible impact when compared to the exact geodesic distance. Note that the exact version performs distance computation as well as local estimation per pixel and is, thus, an order of magnitude slower to compute, see last column of Table 2.

Contour	Distance	MPI-Sintel	Kitti	Middlebury	Time
SED [15]	**Geodesic (approx.)**	3.686	3.334	**0.380**	16.4s
SED [15]	Geodesic (exact)	**3.677**	**3.216**	0.393	204s
-	Euclidean	4.617	3.663	0.442	40s
SED [15]	mixed	3.975	3.510	0.399	300s
gPb [3]	Geodesic (approx.)	4.161	3.437	0.430	26s
Canny [11]	Geodesic (approx.)	4.551	3.308	0.488	16.4s
$\|\nabla_2 \mathcal{I}\|_2$	Geodesic (approx.)	4.061	3.399	0.388	16.4s
GT boundaries	Geodesic (approx.)	3.588			

Table 2. Comparison of the AEE of EpicFlow (with DM and LA) for different distances and different contour extractors. The time (right column) is reported for a MPI-Sintel image pair.

Next, we compare the geodesic distance and Euclidean distances. Table 2 shows that using a Euclidean distance leads to a significant drop in performance, in particular for the MPI-Sintel dataset, the drop is 1 pixel. This confirms the importance of our edge-preserving distance. Note that the result with the Euclidean distance is reported with an exact version, *i.e.*, the interpolation is computed pixelwise.

We also compare to a mixed approach, in which the neighbor list $\mathcal{N}_K$ is constructed using the Euclidean distance, but weights $k_{\tilde{D}}(\boldsymbol{p}_m, \boldsymbol{p})$ are set according to the approximate geodesic distance. Table 2 shows that this leads to a drop of performance by around 0.3 pixels for MPI-Sintel and Kitti. Figure 6 illustrates the reason: none of the Euclidean neighbor matches (yellow) belong to the region corresponding to the selected pixel (red), but all of geodesic neighbor matches (blue) belong to it. This demonstrates the importance of using an edge-preserving geodesic distance throughout the whole pipeline, in contrast to [22] who interpolates matches found in a Euclidean neighborhood.

Impact of contour detector. We also evaluate the impact of the contour detector in Table 2, *i.e.*, the SED detector [15] is replaced by the Berkeley gPb detector [3] or the Canny edge detector [11]. Using gPb leads to a small drop in performance (around 0.1 pixel on Kitti and 0.5 on MPI-Sintel) and significantly increases the computation time. Canny edges perform similar to the Euclidean distance. This can be explained by the insufficient quality of the Canny contours. Using the norm of image's gradient improves slightly over gPb. We found that this is due to the presence of holes when estimating contours with gPb. Finally, we perform experiments using ground-truth motion boundaries, computed from the norm of ground-truth flow gradient, and obtain an improvement of 0.1 on MPI-Sintel (0.2 before the variational part). The ground-truth flow is not dense enough on Middlebury and Kitti datasets to estimate GT boundaries.

5.3. EpicFlow versus coarse-to-fine scheme

To show the benefit of our approach, we have carried out a comparison with a coarse-to-fine scheme. Our implementation of the variational approach is the same as in Section 4, with a coarse-to-fine scheme and DeepMatching integrated in the energy through a penalization of the difference between flow and matches [9, 34]. Table 3 compares EpicFlow to the variational approach with coarse-to-fine scheme, using exactly the same matches as input. EpicFlow performs better and is also faster. The gain is around 0.4 pixel on MPI-Sintel and over 1 pixel on Kitti. The important gain on Kitti might be explained by the affine model used for interpolation, which fits well the piecewise planar structure of the scene. On Middlebury, the variational approach achieves slightly better results, as this dataset does not contain large displacements.

Flow method	MPI-Sintel	Kitti	Middlebury	Time
DM+coarse-to-fine	4.095	4.422	**0. 321**	25s
DM+EpicFlow	**3.686**	**3.334**	0.380	16.4s

Table 3. Comparison of AEE for EpicFlow (with DM + LA) and a coarse-to-fine scheme (with DM).

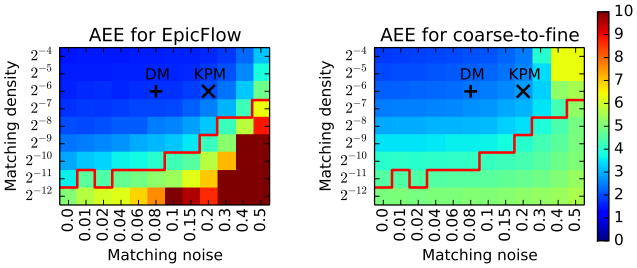

Figure 8. Comparison of AEE between EpicFlow (left) and a coarse-to-fine scheme (right) for various synthetic input matches with different densities and error levels. For positions above the red line, EpicFlow performs better.

Figure 7 shows a comparison to the state-of-the-art method, built upon a coarse-to-fine scheme. Note how motion boundaries are preserved by EpicFlow. Even small details, like the limbs in the right column, are captured.

Sensitivity to the matching quality. In order to get a better understanding of why EpicFlow performs better than a coarse-to-fine scheme, we have evaluated and compared their performances for different densities and error rates of the input matches. To that aim, we generated synthetic matches by taking the ground-truth flow, removing points in the occluded areas, subsampling to obtain the desired density and corrupting the matches to the desired percentage of incorrect matches. For each set of matches with a given density and quality, we have carefully determined the parameters of EpicFlow and the coarse-to-fine method on the MPI-Sintel training subset, and then evaluated them on the remaining training images.

Results in term of AEE are given in Figure 8, where density is represented vertically as the ratio of #matches / #non-occluded pixels and matching error is represented horizontally as the ratio of #false matches / #matches. We can observe that EpicFlow yields better results provided that the matching is sufficiently dense for a given error rate. For low-density or strongly corrupted matches, EpicFlow yields unsatisfactory performance (Figure 8 left), while the coarse-to-fine method remains relatively robust (Figure 8 right). This shows that our interpolation-based heuristic for initial-

Method	AEE	AEE-occ	s0-10	s10-40	s40+	Time
EpicFlow	**6.285**	**32.564**	**1.135**	**3.727**	**38.021**	**16.4s**
TF+OFM [20]	6.727	33.929	1.512	3.765	39.761	~400s
DeepFlow [34]	7.212	38.781	1.284	4.107	44.118	19s
S2D-Matching [22]	7.872	40.093	1.172	4.695	48.782	~2000s
Classic+NLP [28]	8.291	40.925	1.208	5.090	51.162	~800s
MDP-Flow2 [38]	8.445	43.430	1.420	5.449	50.507	709s
NLTGV-SC [25]	8.746	42.242	1.587	4.780	53.860	
LDOF [9]	9.116	42.344	1.485	4.839	57.296	30s

Table 4. Results on MPI-Sintel test set (final version). AEE-occ is the AEE on occluded areas. s0-10 is the AEE for pixels whose motions is between 0 and 10 px and similarly for s10-40 and s40+.

Method	AEE-noc	AEE	Out-Noc 3	Out-All 3	Time
EpicFlow	**1.5**	3.8	7.88%	17.08%	16s
NLTGV-SC [25]	1.6	3.8	**5.93%**	11.96%	16s (GPU)
BTF-ILLUM [14]	**1.5**	**2.8**	6.52%	**11.03%**	80s
TGV2ADCSIFT [7]	**1.5**	4.5	6.20%	15.15%	12s (GPU)
Data-Flow [30]	1.9	5.5	7.11%	14.57%	180s
DeepFlow [34]	**1.5**	5.8	7.22%	17.79%	17s
TF+OFM [20]	2.0	5.0	10.22%	18.46%	350s

Table 5. Results on Kitti test set. AEE-noc is the AEE over non-occluded areas. Out-Noc 3 (resp. Out-all 3) refers to the percentage of pixels where flow estimation has an error above 3 pixels in non-occluded areas (resp. all pixels).

izing the flow takes better advantage of the input matches than a coarse-to-fine schemes for sufficiently dense matches and is able to recover from matching failures. We have indicated the position of DeepMatching and KPM in terms of density and quality on the plots: they lie inside the area in which EpicFlow outperforms a coarse-to-fine scheme.

5.4. Comparison with the state of the art

Results on MPI-Sintel test set are given in Table 4. Parameters are optimized on the MPI-Sintel training set. EpicFlow outperforms the state of the art with a gap of 0.5 pixel in AEE compared to the second best performing method, TF+OFM [20], and 1 pixel compared to the third one, DeepFlow [34]. In particular, we improve for both AEE on occluded areas and AEE over all pixels and for all displacement ranges. In addition, our approach is significantly faster than most of the methods, *e.g.* an order of magnitude faster than the second best.

Table 5 reports the results on the Kitti test set for methods that do not use epipolar geometry or stereo vision. Parameters are optimized on the Kitti training set. We can see that EpicFlow performs best in terms of AEE on non-occluded areas. In term of percentage of erroneous pixels, our method is competitive with the other algorithms. When comparing the methods on both Kitti and MPI-Sintel, we outperform TF+OFM [20] and DeepFlow [34] (second and third on MPI-Sintel) on the Kitti dataset, in particular for occluded areas. We perform on par with NLTGV-SC [25] on Kitti that we outperform by 2.5 pixels on MPI-Sintel.

On the Middlebury test set, we obtain an AEE below 0.4 pixel. This is competitive with the state of the art. In this dataset, there are no large displacements, and consequently,

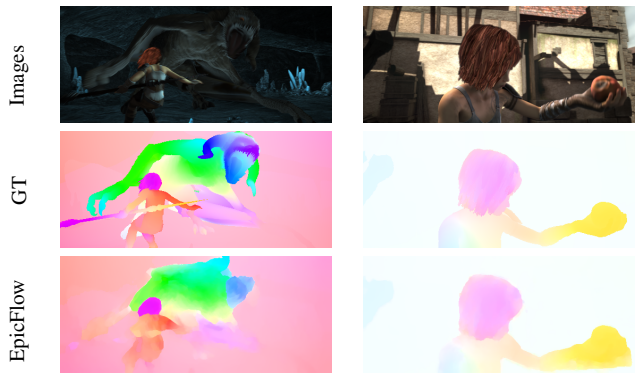

Figure 9. Failure cases of EpicFlow due to missing matches on spear and horns of the dragon (left column) and missing contours on the arm (right column).

the benefits of a matching-based approach are limited. Note that we have slightly increased the number of fixed point iterations to 25 in the variational method for this dataset (still using one level) in order to get an additional smoothing effect. This leads to a gain of 0.1 pixels (measured on the Middlebury training set when setting the parameters on MPI-Sintel training set).

Timings. EpicFlow runs in 16.4 seconds for a MPI-Sintel image pair (1024×436 pixels) on one CPU-core at 3.6Ghz. In detail, computing DeepMatching takes 15s, extracting SED edges 0.15s, dense interpolation 0.25s, and variational minimization 1s.

Failure cases. EpicFlow can be incorrect due to errors in the sparse matches or errors in the contour extraction. Figure 9 (left column) shows an example where matches are missing on thin elements (spear and horns of the dragon). Thus, the optical flow takes the value of the surrounding region for these elements. An example for incorrect contour extraction is presented in Figure 9 (right column). The contour of the character's left arm is poorly detected. As a result, the motion of the arm spreads into the background.

6. Conclusion

This paper introduces EpicFlow, a novel state-of-the-art optical flow estimation method. EpicFlow computes a dense correspondence field by performing a sparse-to-dense interpolation from an initial sparse set of matches, leveraging contour cues using an edge-aware geodesic distance. The approach builds upon the assumption that contours often coincide with motion discontinuities. The resulting dense correspondence field is fed as an initial optical flow estimate to a one-level variational energy minimization. Experimental results show that EpicFlow outperforms current coarse-to-fine approaches. Both the sparse set of matches and the contour estimates are key to our approach. Future work will focus on improving these two components separately as well as in an interleaved manner.

Acknowledgments. This work was supported by the European integrated project AXES, the MSR-Inria joint centre, the LabEx Persyval-Lab (ANR-11-LABX-0025), the Moore-Sloan Data Science Environment at NYU, and the ERC advanced grant ALLEGRO.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. PAMI*, 2012. 2

[2] L. Alvarez, J. Weickert, and J. Sánchez. Reliable estimation of dense optical flow fields with large displacements. *IJCV*, 2000. 2

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. PAMI*, 2011. 7

[4] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 2011. 2, 5

[5] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving patchmatch for large displacement optical flow. *IEEE Trans. Image Processing*, 2014. 2

[6] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized PatchMatch correspondence algorithm. In *ECCV*, 2010. 1, 2, 5

[7] J. Braux-Zin, R. Dupont, and A. Bartoli. A general dense image matching framework combining direct and feature-based costs. In *ICCV*, 2013. 2, 8

[8] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, 2004. 1, 2, 5

[9] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Trans. PAMI*, 2011. 1, 2, 7, 8

[10] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012. 1, 2, 5

[11] J. Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 1986. 7

[12] Z. Chen, H. Jin, Z. Lin, S. Cohen, and Y. Wu. Large displacement optical flow from nearest neighbor fields. In *CVPR*, 2013. 2

[13] A. Criminisi, T. Sharp, C. Rother, and P. Pérez. Geodesic image and video editing. *ACM Trans. Graph.*, 2010. 3

[14] O. Demetz, M. Stoll, S. Volz, J. Weickert, and A. Bruhn. Learning brightness transfer functions for the joint recovery of illumination changes and optical flow. In *ECCV*. 2014. 8

[15] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 1, 2, 3, 4, 7

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *IJRR*, 2013. 2, 5

[17] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. 3

[18] K. He and J. Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *CVPR*, 2012. 2, 5

[19] B. K. P. Horn and B. G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 1981. 1, 2

[20] R. Kennedy and C. J. Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *EMMCVPR*, 2015. 8

[21] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *ECCV*. 2014. 3

[22] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Locally affine sparse-to-dense matching for motion and occlusion estimation. In *ICCV*, 2013. 1, 2, 4, 7, 8

[23] J. Lu, H. Yang, D. Min, and M. Do. Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation. In *CVPR*, 2013. 2

[24] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optic flow computation with theoretically justified warping. *IJCV*, 2006. 1

[25] R. Ranftl, K. Bredies, and T. Pock. Non-local total generalized variation for optical flow estimation. In *ECCV*, 2014. 8

[26] X. Ren. Local grouping for optical flow. In *CVPR*, 2008. 2

[27] F. Steinbrucker, T. Pock, and D. Cremers. Large displacement optical flow computation without warping. In *ICCV*, 2009. 2

[28] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *IJCV*, 2014. 1, 2, 8

[29] D. Sun, E. B. Sudderth, and M. J. Black. Layered image motion with explicit occlusions, temporal consistency, and depth ordering. In *NIPS*, 2010. 2

[30] C. Vogel, S. Roth, and K. Schindler. An evaluation of data costs for optical flow. In *GCPR*, 2013. 8

[31] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2010. 3

[32] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 2008. 3

[33] A. Wedel, D. Cremers, T. Pock, and H. Bischof. Structure- and motion-adaptive regularization for high accuracy optic flow. In *ICCV*, 2009. 5

[34] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *ICCV*, 2013. 1, 2, 4, 5, 6, 7, 8

[35] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *BMVC*, 2009. 1

[36] J. Wills, S. Agarwal, and S. Belongie. What went where. In *CVPR*, 2003. 2

[37] J. Wills, S. Agarwal, and S. Belongie. A feature-based approach for dense segmentation and estimation of large disparity motion. *IJCV*, 2006. 2

[38] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *IEEE Trans. PAMI*, 2012. 1, 2, 5, 8

[39] D. M. Young. *Iterative solution of large linear systems*. Academic Press, 1971. 5

[40] H. Zimmer, A. Bruhn, and J. Weickert. Optic flow in harmony. *IJCV*, 2011. 5