EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow
Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, Cordelia Schmid

To cite this version:
Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, Cordelia Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. 2015. hal-01097477v2

HAL Id: hal-01097477
https://inria.hal.science/hal-01097477v2
Preprint submitted on 27 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow

Jerome Revaud\textsuperscript{a}  Philippe Weinzaepfel\textsuperscript{a}  Zaid Harchaoui\textsuperscript{a,\,b}  Cordelia Schmid\textsuperscript{a}

\textsuperscript{a} Inria \textsuperscript{\ast}  \textsuperscript{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 \cite{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 \textit{i.e.}, less than a few pixels \cite{19, 35, 24}. These approaches cast the optical flow problem into an energy minimization framework, often solved using efficient coarse-to-fine algorithms \cite{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 \cite{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 \cite{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, \textit{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.

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 \cite{6, 34}. In the same spirit as \cite{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: \textit{motion boundaries often tend to appear at image edges}, see Figure 1. Consequently, we propose to exchange the Euclidean distance with a better, \textit{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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image1.png}
\caption{Image edges detected with SED \cite{15} and ground-truth optical flow. Motion discontinuities appear most of the time at image edges.}
\end{figure}
there may be better initialization strategies than the well-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 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 Patch-Match [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 transformation 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

The proposed approach, EpicFlow, consists of three steps, as illustrated in Figure 3. First, we compute a sparse set of matches between the two images, using a state-of-the-
3.2. Interpolation method

The interpolation requires a distance \( D \) between pixels, see Section 3.3. We consider here two options for the interpolation.

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

\[
F_{NW}(p) = \frac{\sum_{(p_m, p_m') \in M} k_D(p_m, p)p_m'}{\sum_{(p_m, p_m') \in M} k_D(p_m, p)}, \quad (1)
\]

where \( k_D(p_m, p) = \exp(-aD(p_m, p)) \) 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 \( F_{LA}(p) \) is interpolated using a locally-weighted affine estimator at a pixel \( p \in I \) as

\[
F_{LA}(p) = A_p p + t_p^T, \quad \text{where } A_p \text{ and } t_p \text{ are parameters of an affine transformation estimated for pixel } p.
\]

These parameters are computed as the least-square solution of an overdetermined system obtained by writing two equations for each match \( (p_m, p'_m) \in M \) weighted as previously:

\[
k_D(p_m, p)(A_p p_m + t_p^T - p_m') = 0 \quad (2)
\]

**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 \( p \) to its \( K \) nearest neighbors according to the distance \( D \), which we denote as \( N_K(p) \). In other words, we replace the summation over \( M \) in the NW operator by a summation over \( N_K(p) \), and likewise for building the overdetermined system to fit the affine transformation for \( F_{LA} \).

3.3. 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. Formally, the geodesic distance between two pixels
Recently to generate object proposals [21]. In practice, we set (denoising, texture flattening, etc.) and it was also used re-
natural tool for edge-preserving image editing operations et al. [32].

number of matches in each region is sufficient. Similarly, motion boundaries
between pixels belonging to the same region will be low. It en-
terpolates similarly, suggesting a strategy that would lever-
are objects/parts of the image with (e-f) geodesic distance from a pixel
1
p and q is defined as the shortest distance with respect to a
cost map C:

\[ D_G(p, q) = \inf_{\Gamma \in \mathcal{P}_{p,q}} \int_\Gamma C(p_s) dp_s, \quad (3) \]

where \( \mathcal{P}_{p,q} \) denotes the set of all possible paths between \( p \)
and \( q \), and \( C(p_s) \) the cost of crossing pixel \( p_s \) (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 inter-
polation will respect the motion boundaries.

In practice, we use an alternative to true motion bound-
aries, making the plausible assumption that image edges are
a superset of motion boundaries. This way, the distance be-
tween pixels belonging to the same region will be low. It en-
sures 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 re-

1https://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 \( \{p_m\} \) from DeepMatching [34]; (e-f) geodesic distance from a pixel \( p \) (marked in blue) to all others \( D_G(p, \cdot) \) (the brighter, the closer). (g-h) 100 nearest matches, i.e., \( N_{100}(p) \) (red) using geodesic distance \( D_G \) from the pixel \( p \) in blue.

3.4. 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 wave-
front. 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 pro-
pose an efficient approximation \( \tilde{D}_G \).

A key observation is that neighboring pixels are often inter-
polated similarly, suggesting a strategy that would leverage
such local information. In this section we employ the
term ‘match’ to refer to \( p_m \) instead of \( (p_m, p_m') \).

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

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

\[ \tilde{D}_G(p, p_m) = D_G(p, L(p)) + D_G^G(L(p), p_m) \quad (4) \]

where \( D_G^G \) is a graph-based approximation of the geodesic
distance between two matches. To define \( D_G^G \) we use a
neighborhood graph \( \mathcal{G} \) whose nodes are \( \{p_m\} \). Two
matches \( p_m \) and \( p_n \) are connected by an edge if they are
neighbors in \( L \). The edge weight is then defined as the
geodesic distance between \( p_m \) and \( p_n \), where the geodesic
distance calculation is restricted to the Voronoi cells of \( p_m \)
and \( p_n \). We, then, calculate the approximate geodesic dis-
tance between any two matches \( p_m, p_n \) using Dijkstra’s al-
gorithm 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 show that it does not impact the accuracy. Let us consider a pixel \( p \) such that \( L(p) = p_m \). The distance between \( p \) and any match \( p_n \) is the same as the one between \( p_m \) and \( p_n \) up to a constant independent from \( p_n \) (Equation 4). As a consequence, we have \( \tilde{N}_K(p) = \tilde{N}_K(p_m) \) and \( k_{DG}(p, p_n) = k_{DG}(p, p_m) \times k_{DG}(p_m, p_n) \). For the Nadaraya-Watson estimator, we thus obtain:

\[
F_{NW}(p) = \frac{\sum_{(p_m, p'_n) \in \tilde{N}_K(p)} k_{DG}(p, p_m) p'_n}{\sum_{(p_m, p'_n)} k_{DG}(p, p_m)}
\]

where all the sums are for \( (p_m, p'_n) \in N_K(p) = N_K(p_m) \). 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 \(|M|\) 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 \( A_{p_m}, t_{p_m} \)” and line 8 “Set \( W_{LA}(p) = A_{L(p)}p + t_{L(p)} \””).

**Algorithm 1** Interpolation with Nadaraya-Watson

**Input:** a pair of images \( I, I' \), a set \( M \) of matches

**Output:** dense correspondence field \( F_{NW} \)

1. Compute the cost \( C \) for \( I \) using SED [15]
2. Compute the assignment map \( L \)
3. Build the graph \( G \) from \( L \)
4. For \( (p_m, p'_n) \in M \)
5. Compute \( \tilde{N}_K(p_m) \) from \( G \) using Dijkstra’s algorithm
6. Compute \( F_{NW}(p_m) \) from \( \tilde{N}_K(p_m) \) using Eq. 1
7. For each pixel \( p \)
8. Set \( F_{NW}(p) = F_{NW}(L(p)) \)

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 \( \alpha \) as in [33, 38]: \( \alpha(x) = \exp \left( -\kappa \|\nabla I(x)\| \right) \) 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 $a \approx 1$ for the coefficient in the kernel $k_D$, the number of neighbors is $K \approx 25$ for NW interpolation and $K \approx 100$ 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\(^3\) 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 sub-sample these pruned correspondences to speed-up the interpolation. We have experimentally verified on several image pairs that this subsampling 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 remove matches for which the difference to the initial estimate is over 5 pixels.

\(\text{http://lear.inrialpes.fr/src/deepmatching/}\)

\(\text{http://josh.github.io/thesis/kdthree/}\)

![Figure 6](image309x668 to 425x719)

**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. All neighbor matches with a Euclidean distance holds to a different object while the geodesic distance allows to capture matches from the same region, even in the case of large occluded areas.

<table>
<thead>
<tr>
<th>Matching</th>
<th>Interpolator</th>
<th>MPI-Sintel</th>
<th>Kitti</th>
<th>Middlebury</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPM</td>
<td>NW</td>
<td>6.052</td>
<td>15.679</td>
<td>0.765</td>
</tr>
<tr>
<td>KPM</td>
<td>LA</td>
<td>6.334</td>
<td>12.011</td>
<td>0.776</td>
</tr>
<tr>
<td>DM</td>
<td>NW</td>
<td>4.143</td>
<td>5.460</td>
<td>0.898</td>
</tr>
<tr>
<td>DM</td>
<td>LA</td>
<td>4.068</td>
<td>3.560</td>
<td>0.840</td>
</tr>
<tr>
<td>KPM</td>
<td>NW</td>
<td>5.741</td>
<td>15.240</td>
<td>0.388</td>
</tr>
<tr>
<td>KPM</td>
<td>LA</td>
<td>5.764</td>
<td>11.307</td>
<td>0.315</td>
</tr>
<tr>
<td>DM</td>
<td>NW</td>
<td>3.804</td>
<td>4.900</td>
<td>0.485</td>
</tr>
<tr>
<td>DM</td>
<td>LA</td>
<td>3.686</td>
<td>3.334</td>
<td>0.380</td>
</tr>
</tbody>
</table>

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

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 examine the impact of the geodesic distance and its approximation as well as the impact 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 3.4.

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 multiscale 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.

The interpolation is robust to the neighborhood size $K$ with for instance an AEE of 4.042, 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 three image pairs with the initialization only and the final result of EpicFlow (row three and four). 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 3.4). 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.

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 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.

Figure 7 shows a comparison to three state-of-the-art methods, all 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. Similarly, in the case of occluded areas, EpicFlow benefits from the geodesic distance to produce a correct estimation, see the right part of the left example.

**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...
Figure 7. Each column shows from top to bottom: mean of two consecutive images, ground-truth flow, result of sparse-to-dense interpolation (Interpolation), full method (EpicFlow), and 3 state-of-the-art methods. EpicFlow better respects motion boundaries, is able to capture small parts like the limbs of the character (right column) and successfully estimates the flow in occluded areas (right part of left column).

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

<table>
<thead>
<tr>
<th>Flow method</th>
<th>MPI-Sintel</th>
<th>Kitti</th>
<th>Middlebury</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM+coarse-to-fine</td>
<td>4.095</td>
<td>4.422</td>
<td>0.321</td>
<td>25s</td>
</tr>
<tr>
<td>DM+EpicFlow</td>
<td>3.686</td>
<td>3.334</td>
<td>0.380</td>
<td>16.4s</td>
</tr>
</tbody>
</table>

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.
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+.

<table>
<thead>
<tr>
<th>Method</th>
<th>AEE</th>
<th>AEE-occ</th>
<th>s0-10</th>
<th>s10-40</th>
<th>s40+</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EpicFlow</td>
<td>6.285</td>
<td>32.564</td>
<td>1.135</td>
<td>3.727</td>
<td>38.021</td>
<td>16.4s</td>
</tr>
<tr>
<td>DeepFlow [34]</td>
<td>7.212</td>
<td>38.781</td>
<td>1.284</td>
<td>4.107</td>
<td>44.118</td>
<td>19s</td>
</tr>
<tr>
<td>S2D-Matching [22]</td>
<td>7.872</td>
<td>40.093</td>
<td>1.172</td>
<td>4.695</td>
<td>48.782</td>
<td>~2000s</td>
</tr>
<tr>
<td>Classic+NLP [23]</td>
<td>8.291</td>
<td>40.925</td>
<td>1.208</td>
<td>5.090</td>
<td>51.162</td>
<td>~8000s</td>
</tr>
<tr>
<td>MDP-Flow2 [35]</td>
<td>8.445</td>
<td>41.430</td>
<td>1.420</td>
<td>5.449</td>
<td>50.507</td>
<td>709s</td>
</tr>
<tr>
<td>LDOF [9]</td>
<td>9.116</td>
<td>42.344</td>
<td>1.485</td>
<td>4.839</td>
<td>57.296</td>
<td>30s</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Method</th>
<th>AEE-noc</th>
<th>AEE</th>
<th>Out-Noc 3</th>
<th>Out-All 3</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EpicFlow</td>
<td>1.2</td>
<td>3.8</td>
<td>7.8%</td>
<td>17.1%</td>
<td>16s</td>
</tr>
<tr>
<td>NLTGV-SC [25]</td>
<td>1.6</td>
<td>3.8</td>
<td>5.9%</td>
<td>11.9%</td>
<td>16s (GPU)</td>
</tr>
<tr>
<td>BTF-ILLUM [14]</td>
<td>1.5</td>
<td>2.8</td>
<td>6.5%</td>
<td>11.0%</td>
<td>80s</td>
</tr>
<tr>
<td>TGV2ADC-SIFT [7]</td>
<td>1.5</td>
<td>4.5</td>
<td>6.2%</td>
<td>15.5%</td>
<td>12s (GPU)</td>
</tr>
<tr>
<td>Data-Flow [10]</td>
<td>1.9</td>
<td>5.5</td>
<td>7.1%</td>
<td>14.5%</td>
<td>180s</td>
</tr>
<tr>
<td>DeepFlow [34]</td>
<td>1.5</td>
<td>5.8</td>
<td>7.2%</td>
<td>17.7%</td>
<td>17s</td>
</tr>
<tr>
<td>TF+OFM [20]</td>
<td>2.0</td>
<td>5.0</td>
<td>10.2%</td>
<td>18.4%</td>
<td>30s</td>
</tr>
</tbody>
</table>

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).
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 project AXES, the MSR-Inria joint centre, the LabEx ALLEGRO.

References


[34] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In ICCV, 2013. 1, 2, 3, 4, 5, 6, 7, 8, 9


[38] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. IEEE Trans. PAMI, 2012. 1, 2, 5, 8, 9
