Fast Optical Flow using Dense Inverse Search

Till Kroeger[1] Radu Timofte[1] Dengxin Dai[1] Luc Van Gool[1,2]
[1]Computer Vision Laboratory, D-ITET, ETH Zurich
[2]VISICS / iMinds, ESAT, K.U. Leuven
{kroegert, timofter, dai, vangool}@vision.ee.ethz.ch

Abstract

Most recent works in optical flow extraction focus on the accuracy and neglect the time complexity. However, in real-life visual applications, such as tracking, activity detection and recognition, the time complexity is critical.

We propose a solution with very low time complexity and competitive accuracy for the computation of dense optical flow. It consists of three parts: 1) inverse search for patch correspondences; 2) dense displacement field creation through patch aggregation along multiple scales; 3) variational refinement. At the core of our Dense Inverse Search-*based method (DIS) is the efficient search of correspondences inspired by the inverse compositional image alignment proposed by Baker and Matthews [3, 1] in 2001.*

DIS is competitive on standard optical flow benchmarks with large displacements. DIS runs at 300Hz up to 600Hz on a single CPU core[1], reaching the temporal resolution of human's biological vision system [8]. It is order(s) of magnitude faster than state-of-the-art methods in the same range of accuracy, making DIS ideal for visual applications.

1. Introduction

Optical flow estimation is under constant pressure to increase both its quality and speed. Such progress allows for new applications. A higher speed enables its inclusion into larger systems with extensive subsequent processing (*e.g*. reliable features for motion segmentation, tracking or action/activity recognition) and its deployment in computationally constrained scenarios (*e.g*. embedded system, autonomous robots, large-scale data processing).

A robust optical flow algorithm should cope with discontinuities (outliers, occlusions, motion discontinuities), appearance changes (illumination, chromacity, blur, deformations), and large displacements. Decades after the pioneering research of Horn and Schunck [27] and Lucas and Kanade [35] we have solutions for the first two issues [9, 38] and recent endeavors lead to significant progress in handling large displacements [45, 13, 11, 31, 49, 28, 36, 41, 2, 51, 53, 54, 55, 18]. This came at the cost of high run-times usually not acceptable in computationally constrained scenarios such as real-time applications. Recently, only very few works aimed at balancing accuracy and run-time in favor of efficiency [17, 48, 54], or employed massively parallelized dedicated hardware to achieve acceptable run-times [4, 40, 18]. In contrast to this, recently it has been noted for several computer vision tasks [24, 44, 8, 6], that it is often desirable to trade-off powerful but complex algorithms for simple and efficients methods, and rely on high frame-rates and smaller search spaces for good accuracy.

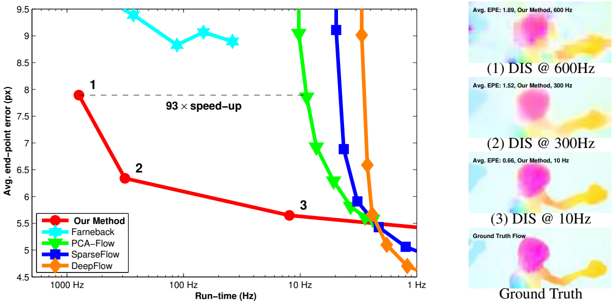


Figure 1: Our DIS method runs at 10Hz up to 600Hz on a single core CPU for an average end-point pixel error smaller or similar to top optical flow methods at comparable speed. This plot excludes preprocessing time for *all* methods. Details in § 3.1,3.3.

In this paper we focus on improving the speed of optical flow in general, non-domain-specific scenarios, while remaining close to the state-of-the-art flow quality. We propose two novel components with low time complexity, one using inverse search for fast patch correspondences, and one based on multi-scale aggregation for fast dense flow estimation. Additionally, a fast variational refinement step further improves the solution of our *dense inverse search*-based method. Altogether, we obtain speed-ups of 1-2 orders of magnitude over state-of-the-art methods at comparable flow quality operating points (Fig. 1). The run-times are in the range of 10-600 Hz on 1024×436 resolution images by us-

[1] 1024×436 resolution. 42Hz / 46Hz when including preprocessing: disk access, image re-scaling gradient computation. More details in § 3.1

ing a single CPU core on a common desktop PC, reaching the temporal resolution of human's biological vision system [8]. To the best of our knowledge, this is the first time that optical flow at several hundred frames-per-second has been reached with such high flow quality on any hardware.

1.1. Related work

Providing an exhaustive overview [20] of optical flow estimation is beyond the scope of this paper. Most of the work on improving the time complexity (without trading-off quality) combines some of the following ideas:

While, initially, the **feature descriptors** of choice were extracted sparsely, invariant under scaling or affine transformations [37], the recent trend in optical flow estimation is to densely extract rigid (square) descriptors from local frames [50, 13, 33]. HOG [15], SIFT [34], and SURF [7] are among the most popular square patch support descriptors. In the context of scene correspondence, the SIFT-flow [33] and PatchMatch [5] algorithms use descriptors or small patches. The descriptors are invariant only to similarities which may be insufficient especially for large displacements and challenging deformations [13].

The **feature matching** usually employs a (reciprocal) nearest neighbor operation [34, 5, 13]. Important exceptions are the recent works of Weinzaepfel *et al.* [51] (non-rigid matching inspired by deep convolutional nets), of Leordeanu *et al.* [31] (enforcing affine constraints), and of Timofte *et al.* [49] (robust matching inspired by compressed sensing). They follow Brox and Malik [13] and guide a variational optical flow estimation through (sparse) correspondences from the descriptor matcher and can thus handle arbitrarily large displacements. Xu *et al.* [55] combine SIFT [34] and PatchMatch [5] matching for refined flow level initialization at the expense of computational costs.

An **optimization** problem is often at the core of the flow extraction methods. The flow is estimated by minimizing an energy that sums up matching errors and smoothness constraints. While Horn and Schunck [27] proposed a variational approach to globally optimize the flow, Lucas and Kanade [35] solve the correspondence problem for each image patch locally and independently. The local [35, 48, 43] methods are usually faster but less accurate than the global ones. Given location and smoothness priors over the image, MRF formulations have been proposed [25, 47].

Parallel computation is a natural way of improving the run-time of the optical flow methods by (re)designing them for parallelization. The industry historically favored specialized hardware such as FPGAs [39], while the recent years brought the advance of GPUs [40, 4, 18, 58]. Yet, multi-core design on the same machine is the most common parallelization. However, many complex flow methods are difficult to adapt for parallel processing.

Learning. Most of the optical flow methods exploit training images for parameter tuning. However, this is only a rough embedding of prior knowledge. Only recently methods were proposed that successfully learn specific models from such training material. Wulff *et al.* [54] assume that any optical flow field can be approximated by a decomposition over a small learned basis of flow fields. Fischer *et al.* [18] construct Convolutional Neural Networks (CNNs) capable to solve the optical flow estimation.

Coarse-to-fine optimizations have been applied frequently to flow estimation [16, 13] to avoid poor local minima, especially for large motions, and thus to improve the performance and to speed up the convergence.

Branch and bound and **Priority queues** have been used to find smart strategies to first explore the flow in the most favorable image regions and gradually refine it for the more ambiguous regions. This often leads to a reduction in computational costs. The PatchMatch method of Barnes *et al.* [5] follows a branch and bound strategy, gradually fixing the most promising correspondences. Bao *et al.* [4] propose the EPPM method based on PatchMatch.

Dynamic Vision Sensors [32], asynchronously capturing illumination changes at microsecond latency, have recently been used to compute optical flow. Benosman [8] and Barranco [6] note that realistic motion estimation tasks, even with large displacements, become simple when capturing image evidence with speed in the kilohertz-range.

1.2. Contributions

We present a novel optical flow method based on dense inverse search (DIS), which we demonstrate to provide high quality flow estimation at 10-600 Hz on a single CPU core. This method is 1-2 orders of magnitude times faster than previous results [51, 49, 54] on the Sintel [14] and KITTI [21] datasets when considering all methods at comparable flow quality operating points. At the same time it is significantly more accurate compared to existing methods running at *equal speed* [17, 35]. This result is based on two main contributions:

Fast inverse search for correspondences. Inspired by the inverse compositional image alignment of Baker and Matthews [3, 1] we devise our inverse search procedure (explained in § 2.1) for fast mining of a grid of patch-based correspondences between two input images. While usually less robust than exhaustive feature matching, we can extract a uniform grid of correspondences in microseconds.

Fast optical flow with multi-scale reasoning. Many methods assume sparse and outlier-free correspondences, and rely heavily on variational refinement to extract pixel-wise flow [51, 49]. This helps to smooth-out small errors, and cover regions with flat and ambigious textures, where exhaustive feature matching often fails. Other methods rely directly on pixel-wise refinement [40, 4]. We chose a middle ground and propose a very fast and robust patch-averaging-scheme, performed only once per scale, after grid-based correspondences have been extracted. This step gains robustness against outlier correspondences, and initializes a pixel-wise variational refinement, performed once per scale. We reach an optimal trade-off between accuracy and speed at 300Hz on a single CPU core, and reach 600Hz without variational refinement at the cost of accuracy. Both operating points are marked as **(2)** and **(1)** in Fig. 1, 4, 5.

Related to our approach is [40]. Here, the inverse image warping idea [1] is used on *all* the pixels, while our method optimizes patches independently. In contrast to our densification done once per scale, they rely on frequent flow interpolations, which requires a high-powered GPU, and still is significantly slower than our CPU-only approach.

The structure of the paper is as follows: In § 2 we introduce our DIS method. In § 3 we describe the experiments, separately evaluate the patch-based correspondence search, and analyse the complete DIS algorithm with and without the variational refinement. In § 4 we conclude the paper.

2. Proposed method

In the following, we introduce our dense inverse search-based method (DIS) by describing: how we extract single point correspondences between two images in § 2.1, how we merge a set of noisy point correspondences on each level s of a scale-pyramid into a dense flow field $\mathbf{U}_s$ in § 2.2, and how we refine $\mathbf{U}_s$ using variational refinement in § 2.3.

2.1. Fast inverse search for correspondences

The core component in our method to achieve high performance is the efficient search for patch correspondences. In the following we will detail how we extract one single point correspondence between two frames.

For a given template patch T in the reference image I_t, with a size of $\theta_{ps} \times \theta_{ps}$ pixels, centered on location $\mathbf{x} = (x,y)^T$, we find the best-matching sub-window of $\theta_{ps} \times \theta_{ps}$ pixels in the query image I_{t+1} using gradient descent. We are interested in finding a warping vector $\mathbf{u} = (u,v)$ such that we minimize the sum of squared differences over the sub-window between template and query location:

$$\mathbf{u} = \text{argmin}_{\mathbf{u}'} \sum_x \left[I_{t+1}(\mathbf{x}+\mathbf{u}') - T(\mathbf{x}) \right]^2 . \qquad (1)$$

Minimizing this quantity is non-linear and is optimized iteratively using the inverse Lukas-Kanade algorithm as proposed in [1]. For this method two steps are alternated for a number of iterations or until the quantity (1) converges. For the first step, the quantity (2) is minimized around the current estimate $\mathbf{u}$ for an update vector $\Delta\mathbf{u}$ such that

$$\Delta\mathbf{u} = \text{argmin}_{\Delta\mathbf{u}'} \sum_x \left[I_{t+1}(\mathbf{x}+\mathbf{u}+\Delta\mathbf{u}') - T(\mathbf{x}) \right]^2 \quad (2)$$

The first step requires extraction and bilinear interpolation of a sub-window $I_{t+1}(\mathbf{x}+\mathbf{u})$ for sub-pixel accurate warp updates. The second step updates the warping $\mathbf{u} \leftarrow \mathbf{u}+\Delta\mathbf{u}$.

Algorithm 1 Dense Inverse Search (DIS)

```
Set initial flow field $\mathbf{U}_{\theta_{ss}+1} \leftarrow \mathbf{0}$
for $s = \theta_{ss}$ to $\theta_{sf}$ do
    (1.) Create uniform grid of $N_s$ patches
    (2.) Initialize displacements from $\mathbf{U}_{s+1}$
    for $i = 1$ to $N_s$ do
        (3.) Inverse search for patch $i$
    (4.) Densification: Compute dense flow field $\mathbf{U}_s$
    (5.) Variational refinement of $\mathbf{U}_s$
```

The original Lukas-Kanade algorithm [35] required expensive re-evaluation of the Hessian of the image warp at every iteration. As proposed in [1] the inverse objective function $\sum_x \left[T(\mathbf{x}-\Delta\mathbf{u}) - I_{t+1}(\mathbf{x}+\mathbf{u}) \right]^2$ can be optimized instead of (2), removing the need to extract the image gradients for $I_{t+1}(\mathbf{x}+\mathbf{u})$ and to re-compute the Jacobian and Hessian at every iteration. Due to the large speed-up this inversion has successfully been used for point tracking in SLAM [29], camera pose estimation [19], and is covered in detail in [1] and our supplementary material.

In order to gain some robustness against absolute illumination changes, we mean-normalize each patch.

One challenge of finding sparse correspondences with this approach is that the true displacements cannot be larger than the patch size θ_{ps}, since the gradient descent is dependent on similar image context in both patches. Often a coarse-to-fine approach with fixed window-size but changing image size is used [29, 19], firstly, to incorporate larger smoothed contexts at coarser scales and thereby lessen the problem of falling into local optima, secondly, to find larger displacements, and, thirdly, to ensure fast convergence.

2.2. Fast optical flow with multi-scale reasoning

We follow this multi-scale approach, but, instead of optimizing patches independently, we compute an intermediate dense flow field and re-initialize patches at each level. This is because of two reasons: 1) the intermediate dense flow field smooths displacements and provides robustness, effectively filtering outliers and 2) it reduces the number of patches on coarser scales, thereby providing a speed-up. We operate in a coarse-to-fine fashion from a first (coarsest) level θ_{ss} in a scale pyramid with a downscaling quotient of θ_{sd} to the last (finest) level θ_{sf}. On each level our method consists of five steps, summarized in algorithm 1, yielding a dense flow field $\mathbf{U}_s$ in each iteration s.

(1.) Creation of a grid: We initialize patches in a uniform grid over the image domain. The grid density and number of patches N_s is implicitly determined by the parameter $\theta_{ov} \in [0,1)$ which specifies the overlap of adjacent patches and is always floored to an integer overlap in pixels. A value of $\theta_{ov} = 0$ denotes a patch adjacency with no overlap and $\theta_{ov} = 1 - \epsilon$ results in a dense grid with one patch centered on each pixel in the reference image.

(2.) Initialization: For the first iteration ($s = \theta_{ss}$) we initialize all patches with the trivial zero flow. On each subsequent scale s we initialize the displacement of each patch $i \in N_s$ at its location $\mathbf{x}$ with the flow from the previous (coarser) scale: $\mathbf{u}_{i,\text{init}} = \mathbf{U}_{s+1}(\mathbf{x}/\theta_{sd}) \cdot \theta_{sd}$.

(3.) Inverse search: Optimal displacements are computed independently for all patches as detailed in § 2.1.

(4.) Densification: After step three we have updated displacement vectors $\mathbf{u}_i$. For more robustness against outliers, we reset all patches to their initial flow $\mathbf{u}_{i,\text{init}}$ for which the displacement update $\|\mathbf{u}_{i,\text{init}} - \mathbf{u}_i\|_2$ exceeds the patch size θ_{ps}. We create a dense flow field $\mathbf{U}_s$ in each pixel $\mathbf{x}$ by applying weighted averaging to displacement estimates of all patches overlapping at $\mathbf{x}$ in the reference image:

$$\mathbf{U}_s(\mathbf{x}) = \frac{1}{Z} \sum_{i}^{N_s} \frac{\lambda_{i,\mathbf{x}}}{\max(1, \|d_i(\mathbf{x})\|_2)} \cdot \mathbf{u}_i \quad , \qquad (3)$$

where the indicator $\lambda_{i,\mathbf{x}} = 1$ iff patch i overlaps with location $\mathbf{x}$ in the reference image, $d_i(\mathbf{x}) = I_{t+1}(\mathbf{x} + \mathbf{u}_i) - T(\mathbf{x})$ denotes the intensity difference between template patch and warped image at this pixel, $\mathbf{u}_i$ denotes the estimated displacement of patch i, and normalization $Z = \sum_i \lambda_{i,\mathbf{x}} / \max(1, \|d_i(\mathbf{x})\|_2)$.

(5.) Variational Refinement: The flow field $\mathbf{U_s}$ is refined using energy minimization as detailed in § 2.3.

2.3. Fast Variational refinement

We use a simplified variant of the variational refinement of [51] without a feature matching term and intensity images only. We refine only on the current scale. The energy is a weighted sum of intensity and gradient data terms (E_I, E_G) and a smoothness term (E_S) over the image domain Ω:

$$E(\mathbf{U}) = \int_{\Omega} \sigma\, \Psi(E_I) + \gamma\, \Psi(E_G) + \alpha\, \Psi(E_S)\, d\mathbf{x} \qquad (4)$$

We use a robust penalizer $\Psi(a^2) = \sqrt{a^2 + \epsilon^2}$, with $\epsilon = 0.001$ for all terms as proposed in [46].

We use a separate penalization of intensity and gradient constancy assumption, with normalization as proposed in [59]: With the brightness constancy assumption $(\nabla_3^T I)\mathbf{u} = 0$, where $\nabla_3 = (\partial x, \partial y, \partial z)^T$ denotes the spatio-temporal gradient, we can model the intensity data term as $E_I = \mathbf{u}^T\, \bar{\mathbf{J}}_0\, \mathbf{u}$. We use the normalized tensor $\bar{\mathbf{J}}_0 = \beta_0\, (\nabla_3 I)(\nabla_3^T I)$ to enforce brightness constancy, with normalization $\beta_0 = (\|\nabla_2 I\|^2 + 0.01)^{-1}$ by the spatial derivatives and a term to avoid division by zero as in [59].

Similarly, E_G penalizes the gradient constancy: $E_G = \mathbf{u}^T\, \bar{\mathbf{J}}_{xy}\, \mathbf{u}$ with $\bar{\mathbf{J}}_{xy} = \beta_x(\nabla_3 I_{dx})(\nabla_3^T I_{dx}) + \beta_y(\nabla_3 I_{dy})(\nabla_3^T I_{dy})$, and normalizations $\beta_x = (\|\nabla_2 I_{dx}\|^2 + 0.01)^{-1}$ and $\beta_y = (\|\nabla_2 I_{dy}\|^2 + 0.01)^{-1}$.

The smoothness term is a penalization over the norm of the gradient of displacements: $E_S = \|\nabla u\|^2 + \|\nabla v\|^2$.

The non-convex energy $E(\mathbf{U})$ is minimized iteratively with θ_{vo} fixed point iterations and θ_{vi} iterations of Successive Over Relaxation (SOR) for the linear system, as in [12].

2.4. Extensions

Our method lends itself to five extensions as follows:

i. Parallelization of all time-sensitive parts of our method (step 3, 5 in § 2.2) is trivially achievable, since patch optimization operates independently, and in the variational refinement the linear systems per pixel are independent as well. We parallelized our implementation with OpenMP and received an almost linear speed-up with number of cores. However, since for fast run-times the overhead of thread creation is significant, we used only a single core in our experiments. Parallelization will be useful in online-scenarios where thread creation is handled once at the start.

ii. Using RGB color images, instead of intensity only, boosts the score in most top-performing optical flow methods. In our experiments, we found that using color is not worth the observed increase of the run-time.

iii. Enforcing forward-backward consistency of flow can boost accuracy. Similarly to using color, we found that the boost is not worth the observed doubling of the run-time.

iv. Robust error norms, such as L1 and the Huber-norm[52], can be used instead of the L2-norm, implicit in the optimization of (1). Experimentally, we found that the gained robustness is not worth the slower convergence.

v. Using DIS for stereo depth, requires the estimation of the horizontal displacement of each pixel. Removing the vertical degree of freedom from our method is trivial.

See the supplementary material for experiments on **i.-v.**

3. Experiments

In order to evaluate the performance of our method, we present three sets of experiments. Firstly, we conduct an analysis of our parameter selection in § 3.1. Here, we also study the impact of variational refinement in our method. Secondly, we evaluate the inverse search (step 3 in algorithm 1) in § 3.2 without densification (step 4). The complete pipeline for optical flow is evaluated in § 3.3, and 3.4. Thirdly, since the problem of recovering large displacements can also be handled by higher frame-rates combined with lower run-time per frame-pair, we conduct an experiment in § 3.5 to analyse the benefit of higher frame-rates.

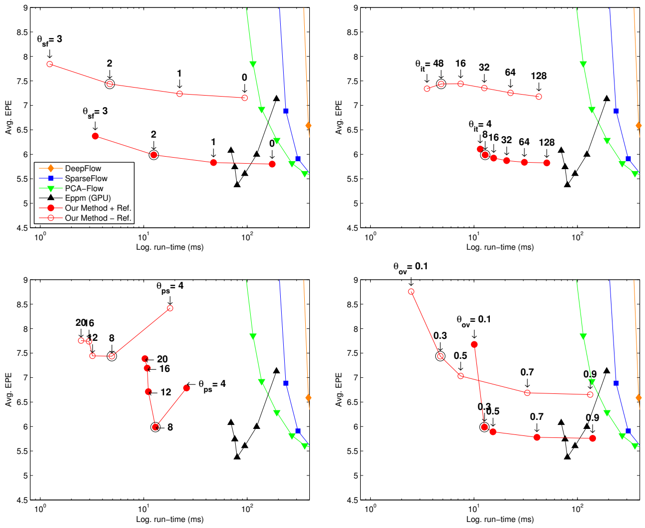


Figure 2: Optical Flow result on Sintel with changing parameters. We set $\theta_{sf} = 2$, $\theta_{it} = 8$, $\theta_{ps} = 8$, $\theta_{ov} = 0.3$, marked with a black circle in all plots. From the top left to bottom right we vary the parameters θ_{sf}, θ_{it}, θ_{ps}, and θ_{ov} independently in each plot.

Parameter	Function
$\theta_{\mathbf{sf}}$	**finest scale in multi-scale pyramide**
$\theta_{\mathbf{it}}$	**number of gradient descent iterations per patch**
$\theta_{\mathbf{ps}}$	**rectangular patch size in (pixel)**
$\theta_{\mathbf{ov}}$	**patch overlap on each scale (percent)**
θ_{sd}	downscaling quotient in scale pyramid
θ_{ss}	coarsest scale in multi-scale pyramid
θ_{vo}, θ_{vi}	number of outer and inner iterations for variational refinement
δ, γ, α	intensity, gradient and smoothness weights for variational refinement

Table 1: Parameters of our method. Parameters in **bold** have a significant impact on performance and are cross-validated in § 3.1.

	EPE all	s0-10	s10-40	s40+
NN	32.06	13.64	53.77	101.00
DIS w/o Densification	7.76	2.16	8.65	37.94
DIS	4.16	**0.84**	4.98	23.09
DeepMatching [51]	**3.60**	1.27	**3.91**	**16.49**

Table 2: Error of sparse correspondences (pixels). Columns left to right: i) average end-point error over complete flow field, ii) error in displacement range < 10 px., iii) $10 - 40$ px., iv) > 40 px.

3.1. Implementation and Parameter Selection

We implemented[2] our method in C++ and run all experiments and baselines on a Core i7 CPU using a single core, and a GTX780 GPU for the EPPM [4] baseline. For *all* experiments on the Sintel and KITTI training datasets we report timings from which we exclude all operations which, in a typical robotics vision application, would be unnecessary, performed only once, or shared between multiple tasks: Disk access, creation of an image pyramid including image gradients with a downsampling quotient of 2, all initializations of the flow algorithms. We do this for our method and *all* baselines within their provided code. For EPPM, where only an executable was available, we subtracted the average overhead time of our method for fair comparison. Please see the supplementary material for variants of these experiments where preprocessing times are included for all methods. Our method requires 20 ms of preprocessing, spent on disk access (11 ms), image scaling and gradients (9 ms). For experiments on the Sintel and KITTI test datasets we include the preprocessing time to be comparable with reported timings in the online benchmarks.

Parameter selection. Our method has four main parameters which affect speed and performance as explained in § 2: θ_{ps} size of each rectangular patch, θ_{ov} patch overlap, θ_{it} number of iterations for the inverse search, θ_{sf} finest and final scale on which to compute the flow. We plot the change in the *average end-point error* versus *run-time* on the Sintel (*training, final*) dataset [14] in Fig. 2. We draw three conclusions: Firstly, operating on finer scales (lower θ_{sf}), more patch iterations (higher θ_{it}), higher patch density (higher θ_{ov}) generally lowers the error, but, depending on the time budget, may not be worth it. Secondly, the patch size θ_{ps} has a clear optimum at 8 and 12 pixels. This also did not change when varying θ_{ps} at lower θ_{sf} or higher θ_{it}. Thirdly, using variational refinement always significantly reduced the error for a moderate increase in run-time.

More implementation details and timings of all parts of algorithm 1 can be found in the supplementary material.

In addition we have several parameters of lower importance, which are fixed for all experiments. We set $\theta_{sd} = 2$, *i.e.* we use a standard image pyramid, where the resolution is halved with each downscaling. We set the coarsest image scale $\theta_{ss} = 5$ for § 3.3 and $\theta_{ss} = 6$ for § 3.4 due to higher image resolutions. For different patch sizes and image pyramids the coarsest scale can be selected as $\theta_{ss} = \log_{\theta_{sd}}(2 \cdot width)/(f \cdot \theta_{ps})$ and raised to the nearest integer, to capture motions of at least $1/f$ of the image width. For the variational refinement we fix intensity, gradient and smoothness weights as $\delta = 5, \gamma = 10, \alpha = 10$ and keep iteration numbers fixed at $\theta_{vo} = 1 \cdot (s + 1)$, where s denotes the current scale and $\theta_{vi} = 5$. In contrast to our comparison baselines [51, 49, 54], we do not fine-tune our method for a specific dataset in our experiments. We use a 20 percent subset of Sintel training to develop our method, and only the remaining training material is used for evaluation. All parameters are summarized in Table 1.

If the flow is not computed up to finest scale ($\theta_{sf} = 0$), we scale-up the result (linearly interpolated) to full resolution for comparison for all methods.

3.2. Evaluation of Inverse Search

In this section we evaluate the sparse point correspondences created by inverse search on the Sintel training

[2] The code will be publicly available on the main author's website.

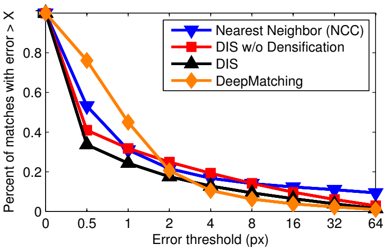


Figure 3: Percent of sparse correspondences above error threshold.

	EPE all	s0-10	s10-40	s40+	time (s)	CPU	GPU
FlowFields [2]	**5.81**	1.16	**3.74**	**33.89**	18 †	✓	
DeepFlow [51]	7.21	1.28	4.11	44.12	55	✓	
SparseFlow [49]	7.85	**1.07**	3.77	51.35	16	✓	
EPPM [4]	8.38	1.83	4.96	49.08	0.31		✓
PCA-Flow [54]	8.65	1.96	4.52	51.84	0.37	✓	
LDOF [13]	9.12	1.49	4.84	57.30	60 † ‡	✓	
Classic+NL-fast [46]	10.09	1.09	4.67	67.81	120 † ‡	✓	
DIS-Fast	10.13	2.17	5.93	59.70	**0.023**	✓	
SimpleFlow [48]	13.36	1.48	9.58	81.35	1.6 † ‡		✓

Table 3: Sintel test errors in pixels (`http://sintel.is.tue.mpg.de/results`), retrieved on 31st Oct. 2015 for *final* subset. Run-times are measured by us, except: †self-reported, and ‡on other datasets with same or smaller resolution.

dataset. For each frame pair we initialized a sparse grid (given by Deep Matching [51]) in the first image and computed point correspondences in the second image. The correspondences are computed by i) exhaustive *Nearest Neighbor* search on normalized cross-correlation (*NCC*), ii) our method where we skip the densification step between each scale change (*DIS w/o Densification*), iii) our method including the densification step (*DIS*), and using iv) *DeepMatching* [51]. The results are shown in Fig. 3 and Table 2.

We have four observations: i) Nearest Neighbor search has a low number of incorrect matches and precise correspondences, but is very prone to outliers. ii) DeepMatching has a high percentage of erroneous correspondences (with small errors), but are very good at large displacements. iii) In contrast to this, our method (DIS w/o Densification) generally produces fewer correspondences with small errors, but is strongly affected by outliers. This is due to the fact that the implicit SSD (sum of squared differences) error minimization is not invariant to changes in orientation, contrast, and deformations. iv) Averaging all patches in each scale (DIS), taking into account their photometric error as described in eq. (3), introduces robustness towards these outliers. It also decreases the error for approximately correct matches. Furthermore, it enables reducing the number of patches at coarser scales, leading to lower run-time.

3.3. MPI Sintel optical flow results

Based on our observations on parameter selection in § 3.1, we selected four operating points, corresponding to

(1) $\theta_{sf} = 3, \theta_{it} = 016, \theta_{ps} = 08, \theta_{ov} = 0.30$, **at 600 Hz**,

(2) $\theta_{sf} = 3, \theta_{it} = 012, \theta_{ps} = 08, \theta_{ov} = 0.40$, **at 300 Hz**,

(3) $\theta_{sf} = 1, \theta_{it} = 016, \theta_{ps} = 12, \theta_{ov} = 0.75$, **at 10 Hz**,

(4) $\theta_{sf} = 0, \theta_{it} = 256, \theta_{ps} = 12, \theta_{ov} = 0.75$, **at 0.5 Hz**.

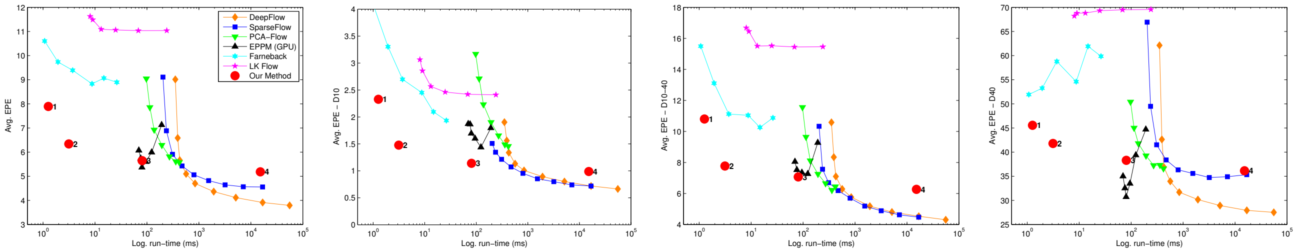
All operating points except **(1)** use variational refinement. We compare our method against a set of recently published baselines running on a single CPU core: DeepFlow [51], SparseFlow [49], PCA-Flow [54]; two older established methods: Pyramidal Lukas-Kanade Flow [10, 35], Farneback's method [17]; and one recent GPU-based method: EPPM [4]. Since run-times for optical flow methods are strongly linked to image resolution, we incrementally speed-up all baselines by downscaling the input images by factor of 2^n, where n starting at $n = 0$ is increased in increments of 0.5. We chose this *non-intrusive* parameter of image resolution to analyse each method's trade-off between run-time and flow error. We bilinearly interpolate the resulting flow field to the original resolution for evaluation.

We run all baselines and our method for all operating points on the Sintel [14] *final* training (Fig. 4) and testing (Table 3) benchmark. As noted in § 3.1, run-times for all methods are reported without preprocessing for the training dataset to facilitate comparison of flow algorithms running in the same environment at high speed, and with preprocessing for the online testing benchmark to allow comparison with self-reported times. From the experiments on the testing and training dataset, we draw several conclusions: *Operating point* **(2)** points to the best trade-off between run-time and flow error. For the average end-point error of around 6 pixels, our method is approximately two orders of magnitude faster than the fastest CPU baseline (PCA-Flow [54]) a still more than one order of magnitude faster than the fastest GPU baseline (EPPM [4]). Our method can be further sped-up by removing the variational refinement at *operating point* **(1)** while maintaining reasonable flow quality (see Fig. 6). *Operating point* **(3)** is comparable with the performance of EPPM, but slightly better for small displacements and worse for large displacements. If we use all available scales, and increase the number of iterations, we obtain *operating point* **(4)**. At the run-time of several seconds per frame pair, more complex methods, such as DeepFlow, perform better, in particular for large displacements.

In the supplementary material we show variants of Fig. 4, and 5 where all preprocessing times are included. Furthermore, we provide flow error maps on Sintel, where typical failure cases of our method at motion discontinuities, large displacements, and frame boundaries are observable.

3.4. KITTI optical flow results

Complementary to the experiment on the synthetic Sintel dataset, we ran our method on the KITTI Optical Flow benchmark [21] for realistic driving scenarios. The result is



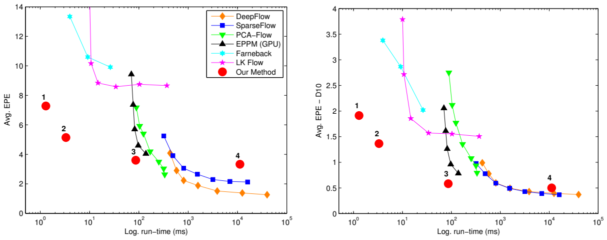
EPE, full range.

EPE for < 10 px.

EPE for 10 − 40 px.

EPE for > 40 px.

Figure 4: Sintel (training) flow result. Average end-point error (pixel) versus run-time (millisecond) on various displacement ranges.



EPE, full range.

EPE for < 10 px.

Figure 5: KITTI (training) result. Average end-point error (px) versus run-time (ms) for average (left) and small displacements (right). See supplementary material for large displacement errors.

	Out-Noc	Out-All	Avg-Noc	Avg-All	time (s)	CPU	GPU
PH-Flow [57]	**5.76 %**	**10.57 %**	**1.3 px**	**2.9 px**	800	✓	
DeepFlow [51]	7.22 %	17.79 %	1.5 px	5.8 px	17	✓	
SparseFlow [49]	9.09 %	19.32 %	2.6 px	7.6 px	10	✓	
EPPM [4]	12.75 %	23.55 %	2.5 px	9.2 px	0.25		✓
PCA-Flow [54]	15.67 %	24.59 %	2.7 px	6.2 px	0.19	✓	
eFolki [40]	19.31 %	28.79 %	5.2 px	10.9 px	0.026		✓
LDOF [13]	21.93 %	31.39 %	5.6 px	12.4 px	60	✓	
FlowNetS+ft [18]	37.05 %	44.49 %	5.0 px	9.1 px	0.08		✓
DIS-Fast	38.58 %	46.21 %	7.8 px	14.4 px	**0.024**	✓	
RLOF [43]	38.60 %	46.13 %	8.7 px	16.5 px	0.488		✓

Table 4: KITTI test results (http://www.cvlibs.net/datasets/kitti/eval_flow.php), retrieved on 31st Oct. 2015, for *all pixels, at 3px threshold*.

presented in fig. 5, 7 (training) and Table 4 (testing). We use the same four operating points as in § 3.3. Our conclusions from the Sintel dataset in § 3.3 also apply for this dataset, suggesting a stable performance of our method, since we did not optimize any parameters for this dataset. On the online test benchmark we are on par with RLOF [43] and the recently published FlowNet [18]. Even though both take advantage of a GPU, we are still faster by one order of magnitude at comparable performance.

We include a plot of more operating points on the training set of Sintel and KITTI in the supplementary material.

3.5. High frame-rate optical flow

Often, a simpler and faster algorithm, combined with a higher temporal resolution in the data, can yield better accuracy than a more powerful algorithm, on lower temporal resolutions. This has been analysed in detail in [24] for the task of visual odometry. As noted in [8, 6] this is also the case for optical flow, where large displacements, due to low-frame rate or strong motions are significantly more difficult to estimate than small displacements. In contrast to the recent focus on handling ever larger displacements [13, 55, 51, 49, 36], we want to analyse how *decreasing* the run-time while *increasing* the frame-rate affects our algorithm. For this experiment we selected a random subset of the Sintel training dataset, and synthesized new ground

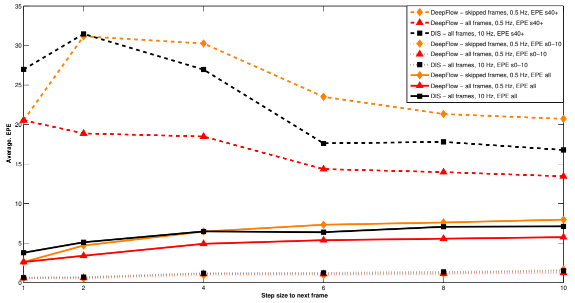


Figure 8: Flow result on Sintel with low temporal resolution. Accuracy of DeepFlow on large displacements versus DIS on small displacements, tracked through *all intermediate* frames. As baseline we included the accuracy of DeepFlow for tracking small displacements. Note: While we use the same frame pairs to compute each vertical set of points, frame pairs differ over stepsizes.

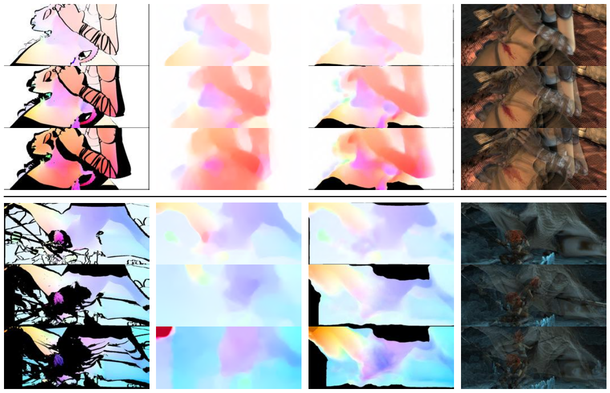


Figure 9: Optical flow on Sintel with lower temporal resolution. In each block of 3x4: Rows, top to bottom, correspond to step sizes 1 (original frame-rate), 6, 10 frames. Columns, left to right, correspond to new ground truth, DeepFlow result, DIS result (through *all intermediate frames*), original images. Large displacements are significantly better preserved by DIS through higher frame-rates.

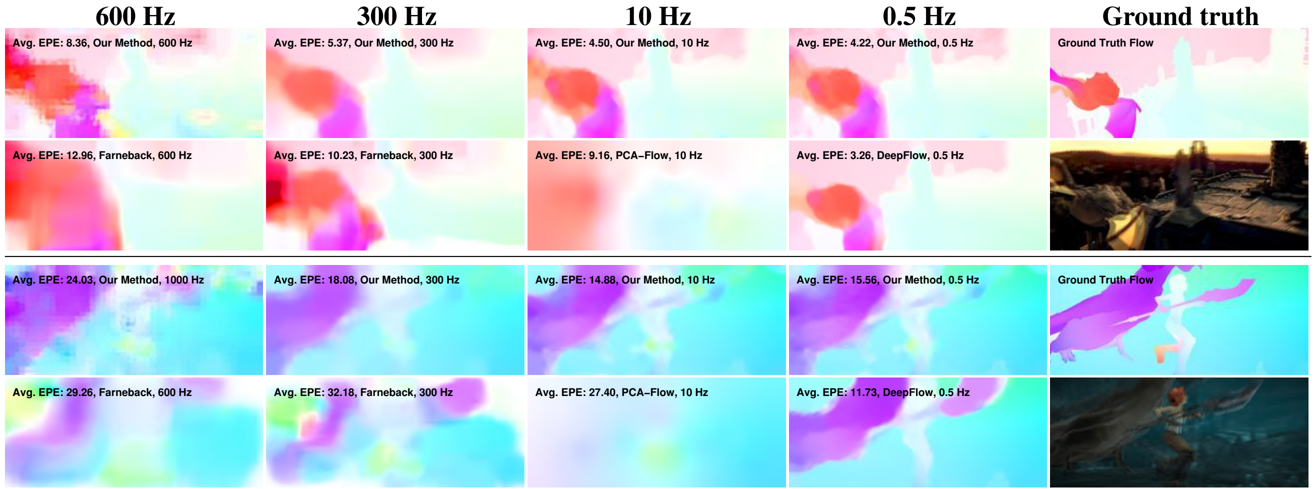


Figure 6: Examplary results on Sintel (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback 600Hz, Farneback 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image.

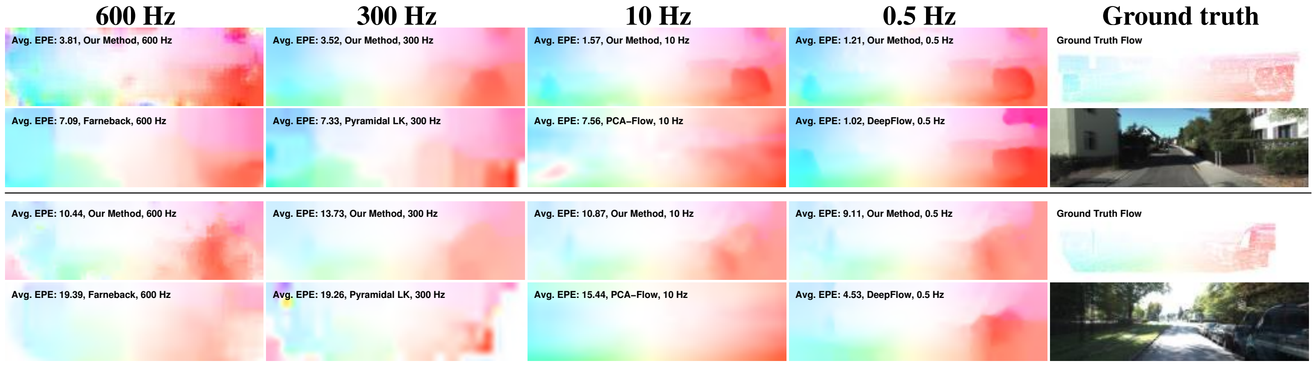


Figure 7: Examplary results on KITTI (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback, 600 Hz, Pyramidal LK 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image.

truth flow for lower frame-rates from the one provided in the dataset. We create new ground truth for $1/2$ to $1/10$ of the source frame-rate from the original ground truth and the additionally provided segmentation masks to invalidate occluded regions. We compare DeepFlow at a speed of 0.5Hz on this lower temporal resolution against our method (*operating point* **(3)**, 10 Hz), running through *all intermediate frames* at the original, higher frame-rate. Thus, while DeepFlow has to handle larger displacements in one frame pair, our method has to handle smaller displacements, tracked through multiple frames and accumulates error drift.

We observe (Fig. 8) that DIS starts to outperform DeepFlow when running at twice the original frame-rate, notably for large displacements, while still being 10 times faster. Fig. 9 shows examples of the new ground truth, results of DeepFlow and DIS. We conclude, that it is advantageous to choose our method over DeepFlow, aimed at recovering large displacements, when the combination of video frame-rate and run-time per frame can be chosen freely.

4. Conclusions

In this paper we presented a novel and simple way of computing dense optical flow. The presented approach trades off a lower flow estimation error for large decreases in run-time: For the same level of error, the presented method is two orders of magnitude faster than current state-of-the-art approaches, as shown in experiments on synthetic (Sintel) and realistic (KITTI) optical flow benchmarks.

In the future we will address some open problems with our method: Due to the coarse-to-fine approach small and fast motions can sometimes get lost beyond recovery. A sampling-based approach to recover over-smoothed object motions at finer scales may alleviate this problem. The implicit minimization of the L2 matching error in our method is not invariant to many modes of change, such as in contrast, deformations, and occlusions. More robust error metrics may be helpful here. Furthermore, a GPU implementation may yield another significant speed-up.

Acknowledgments: This work was supported by the European Research Council project *VarCity (#273940)*. We thank Richard Hartley for his pertinent input on this work.

References

[1] Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004. 1, 2, 3, 10

[2] C. Bailer, B. Taetz, and D. Stricker. Flow fields:dense correspondence fields for highly accurate large displacement optical flow estimation. In *ICCV*, 2015. 1, 6

[3] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *CVPR*, 2001. 1, 2, 10

[4] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving patchmatch for large displacement optical flow. *Image Processing, IEEE Transactions on*, 23(12):4996–5006, Dec 2014. 1, 2, 5, 6, 7

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

[6] F. Barranco, C. Fermuller, and Y. Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proceedings of the IEEE*, 102(10):1537–1556, 2014. 1, 2, 7

[7] H. Baya, A. Essa, T. Tuytelaarsb, and L. Van Gool. Speeded-up robust features (surf). *CVIU*, 110(3):346–359, 2008. 2

[8] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. Event-based visual flow. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(2):407–417, 2014. 1, 2, 7

[9] M. J. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *CVIU*, 1996. 1

[10] J.-Y. Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5:1–10, 2001. 6

[11] J. Braux-Zin, R. Dupont, and A. Bartoli. A general dense image matching framework combining direct and feature-based costs. In *ICCV*, 2013. 1

[12] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, 2004. 4

[13] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Trans. PAMI*, 2011. 1, 2, 6, 7

[14] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012. 2, 5, 6, 13

[15] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 2

[16] W. Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, 43:150–177, 1988. 2

[17] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*, volume 2749 of *Lecture Notes in Computer Science*, pages 363–370. 2003. 1, 2, 6

[18] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 1, 2, 7

[19] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. *ICRA*, pages 15–22, may 2014. 3

[20] D. Fortun, P. Bouthemy, and C. Kervrann. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 134:1 – 21, 2015. Image Understanding for Real-world Distributed Video Networks. 2

[21] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *IJRR*, 2013. 2, 6, 13

[22] A. Geiger, M. Roser, and R. Urtasun. Efficient large-scale stereo matching. In *ACCV*, ACCV, 2011. 13

[23] G. D. Hager and P. N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *TPAMI*, 20(10):1025–1039, 1998. 10

[24] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. Real-time camera tracking: When is high frame-rate best? In *ECCV*, pages 222–235. 2012. 1, 7

[25] F. Heitz and P. Bouthemy. Multimodal estimation of discontinuous optical flow using markov random fields. *TPAMI*, 15(12):1217–1232, Dec 1993. 2

[26] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *TPAMI*, 30(2):328–341, 2008. 13

[27] B. K. Horn and B. G. Schunck. Determining optical flow. *Proc. SPIE 0281, Techniques and Applications of Image Understanding*, 1981. 1, 2

[28] R. Kennedy and C. Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 8932 of *Lecture Notes in Computer Science*, pages 364–377. 2015. 1

[29] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. *ISMAR*, 2007. 3

[30] K. Konolige. Small vision systems: Hardware and implementation. In *Robotics Research*, pages 203–212. Springer London, 1998. 13

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

[32] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, 43(2):566–576, 2008. 2

[33] C. Liu, J. Yuen, and A. Torralba. SIFT flow: Dense correspondence across scenes and its applications. *TPAMI*, 2011. 2

[34] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 2

[35] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981. 1, 2, 3, 6

[36] M. Menze, C. Heipke, and A. Geiger. In *Pattern Recognition*, volume 9358 of *Lecture Notes in Computer Science*, pages 16–28. 2015. 1, 7

[37] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 2005. 2

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

[39] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. Van Hulle. A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features. *Computers, IEEE Transactions on*, 61(7):999–1012, 2012. 2

[40] A. Plyer, G. Le Besnerais, and F. Champagnat. Massively parallel lucas kanade optical flow for real-time video processing applications. *Journal of Real-Time Image Processing*, pages 1–18, 2014. 1, 2, 3, 7

[41] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. EpicFlow:Edge-Preserving Interpolation of Correspondences for Optical Flow. In *CVPR*, 2015. 1

[42] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *GCPR*. 2014. 13

[43] T. Senst, V. Eiselein, and T. Sikora. Robust local optical flow for feature tracking. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(9):1377–1387, 2012. 2, 7

[44] N. Srinivasan, R. Roberts, and F. Dellaert. High frame rate egomotion estimation. In *ICVS*, ICVS'13, pages 183–192, 2013. 1

[45] F. Steinbrucker, T. Pock, and D. Cremers. Large displacement optical flow computation without warping. In *ICCV*, 2009. 1

[46] D. Sun, S. Roth, and M. J. Black. Secrets of optical flow estimation and their principles. In *CVPR*, 2010. 4, 6

[47] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *TPAMI*, 30(6):1068–1080, June 2008. 2

[48] M. Tao, J. Bai, P. Kohli, and S. Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. In *Computer Graphics Forum*, volume 31, pages 345–353. Wiley Online Library, 2012. 1, 2, 6

[49] R. Timofte and L. Van Gool. Sparseflow: Sparse matching for small to large displacement optical flow. In *WACV*, pages 1100–1106, Jan 2015. 1, 2, 5, 6, 7

[50] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. In *CVPR*, 2008. 2

[51] 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, 11

[52] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *BMVC*, 2009. 4

[53] J. Wills, S. Agarwal, and S. Belongie. A feature-based approach for dense segmentation and estimation of large disparity motion. *IJCV*, 2006. 1

[54] J. Wulff and M. J. Black. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. In *CVPR*, pages 120–130, 2015. 1, 2, 5, 6, 7

[55] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *IEEE Trans. PAMI*, 2012. 1, 2, 7

[56] K. Yamaguchi, D. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *ECCV*. 2014. 13

[57] J. Yang and H. Li. Dense, accurate optical flow estimation with piecewise parametric model. In *CVPR*, pages 1019–1027, 2015. 7

[58] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *In Ann. Symp. German Association Patt. Recogn*, 2007. 2

[59] H. Zimmer, A. Bruhn, and J. Weickert. Optic flow in harmony. *IJCV*, 2011. 4

Supplementary Material

A. Derivation of the *fast inverse search* in § 2.1 of the paper

We adopt the terminology of [3, 1] and closely follow their derivation. We consider $\mathbf{W}(\mathbf{x};\mathbf{u})$ a warp, parametrized by $\mathbf{u} = (u,v)^T$, on pixel $\mathbf{x}$ such that $\mathbf{W}(\mathbf{x};\mathbf{u}) = (x+u, y+v)$. The following derivation holds for other warps as well: See [1] for a discussion on the limits of its applicability. The objective function for the inverse search, eq. (1) in the paper, then becomes

$$\sum_x \left[I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u})) - T(\mathbf{x})\right]^2 \; . \tag{5}$$

The warp parameter $\mathbf{u}$ is found by iteratively minimizing

$$\sum_x \left[I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u}+\mathbf{\Delta u})) - T(\mathbf{x})\right]^2 \tag{6}$$

and updating the warp parameters as $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{\Delta u}$.

A Gauss-Newton gradient descent minimization is used in the following. We use a first-order Taylor expansion of eq. (6) on $I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u}+\mathbf{\Delta u}))$:

$$\sum_x \left[I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u})) + \nabla I_{t+1}\frac{\partial \mathbf{W}}{\partial \mathbf{u}}\mathbf{\Delta u} - T(\mathbf{x})\right]^2 \tag{7}$$

where ∇I_{t+1} is the image gradient at $\mathbf{W}(\mathbf{x};\mathbf{u})$. $\frac{\partial \mathbf{W}}{\partial \mathbf{u}}$ denotes the Jacobian of the warp. Writing the partial derivatives in $\frac{\partial \mathbf{W}}{\partial \mathbf{u}}$ with respect to a column vector as row vectors, this simply becomes the 2×2 identity matrix for the case of optical flow.

There is a closed-form solution for parameter update $\mathbf{\Delta u}$ using a least-squares formulation. Setting to zero the partial derivative of eq. (7) with respect to $\mathbf{\Delta u}$

$$\sum_x S^T \cdot \left[I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u})) + \nabla I_{t+1}\frac{\partial \mathbf{W}}{\partial \mathbf{u}}\mathbf{\Delta u} - T(\mathbf{x})\right] \overset{!}{=} 0 \, , \tag{8}$$

where $S = \left[\nabla I_{t+1} \frac{\partial \mathbf{W}}{\partial \mathbf{u}}\right]$, lets us solve for $\mathbf{\Delta u}$ as

$$\mathbf{\Delta u} = H^{-1} \sum_x S^T \cdot \left[T(\mathbf{x}) - I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u}))\right] \tag{9}$$

where $H = \sum_x S^T S$ is the $n \times n$ approximation to the Hessian matrix.

Since S depends on the image gradient of image I_{t+1} at the displacement $\mathbf{u}$, S and the Hessian H have to be reevaluated at each iteration. In order to avoid this costly reevaluation it was proposed [3, 1, 23] to invert to roles of image and template. As a result, the objective function becomes

$$\sum_x \left[T(\mathbf{W}(\mathbf{x};\mathbf{\Delta u})) - I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u}))\right]^2 \; . \tag{10}$$

The new warp is now composed using the *inverse* updated warp $\mathbf{W}(\mathbf{x};\mathbf{u}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{u}) \circ \mathbf{W}(\mathbf{x};\mathbf{\Delta u})^{-1}$. In the case of optical flow, this simply becomes: $\mathbf{u} \leftarrow \mathbf{u} - \mathbf{\Delta u}$.

The first order Taylor expansion of (10) gives

$$\sum_x \left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{u}}\mathbf{\Delta u} - I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u})\right]^2 \tag{11}$$

where $W(x;\mathbf{0})$ is the identity warp. Analogously to eq. (9) we can solve (11) as a least squares problem.

$$\mathbf{\Delta u} = H'^{-1} \sum_x S'^T \cdot \left[I_{t+1}(\mathbf{W}(\mathbf{x};\mathbf{u})) - T(\mathbf{x})\right] \tag{12}$$

where $S' = \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{u}}\right]$ and $H' = \sum_x S'^T S'$. The Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{u}}$ is evaluated at $(\mathbf{x};\mathbf{0})$. Since neither S', nor H' depend in $\mathbf{u}$, they can be pre-computed. This also means that image gradients do not have to be updated in each iteration. In each iteration we only need, firstly, to compute the sum in (12), which includes the image difference, multiplication with the image gradients, and summation, secondly, to solve the linear system for $\mathbf{\Delta u}$, and, thirdly, update the warp parameter $\mathbf{u}$.

B. Extensions: Color, forward-backward consistency, robust error norms

We examined the benefit of, firstly, using RGB color images instead of intensity images, secondly, enforcing forward-backward consistency of optical flow, and, thirdly, using robust error norms.

(i) Using RGB color images. Instead of using a patch matching error over intensity images, we can use a multi-channel image for the dense inverse search and the variational refinement. The objective function for dense the inverse search, eq. (1) in the paper, becomes

$$\sum_{c \in [R,G,B]} \sum_{x} \left[I^c_{t+1}(\mathbf{x}+\mathbf{u}) - T^c(\mathbf{x}) \right]^2 . \tag{13}$$

where c iterates over all color channels. Similarly, we can extend the variational refinement to operate on all color channels jointly, as detailed in [51].

(ii) Enforcing forward-backward consistency. In order to enforce forward-backward consistency, we run our algorithm in parallel from both directions: $I_t \rightarrow I_{t+1}$ and $I_{t+1} \rightarrow I_t$. With the exception of the densification (step 4), all steps of the algorithm run independently for forward and backward flow computation. In step 4 we merge both directions and create a dense forward flow field $\mathbf{U}^f_s$ in each pixel $\mathbf{x}$ by applying weighted averaging to displacement estimates of all patches *from the forward and backward inverse search* overlapping at location $\mathbf{x}$ in the reference image I_t:

$$\mathbf{U}^f_s(\mathbf{x}) = \frac{1}{Z}\left[\sum_{i}^{N^f_s} \frac{\lambda^f_{i,\mathbf{x}}}{\max(1, \|d^f_i(\mathbf{x})\|_2)} \cdot \mathbf{u}^f_i - \right. \tag{14}$$

$$\left. \sum_{j}^{N^b_s} \frac{\lambda^b_{j,\mathbf{x}}}{\max(1, \|d^b_j(\mathbf{x})\|_2)} \cdot \mathbf{u}^b_j \right] \tag{15}$$

where the indicator $\lambda^f_{i,\mathbf{x}} = 1$ iff patch i for the *forward* displacement estimate overlaps with location $\mathbf{x}$ in the reference image, $\lambda^b_{j,\mathbf{x}} = 1$ iff patch j for the *backward* displacement estimate overlaps with location $\mathbf{x}$ in the reference image after the displacement $\mathbf{u}^f_j$ was applied to it, $d^f_i(\mathbf{x}) = I_{t+1}(\mathbf{x} + \mathbf{u}^f_i) - T(\mathbf{x})$ and $d^b_j(\mathbf{x}) = I_t(\mathbf{x}) - T(\mathbf{x} - \mathbf{u}^b_j)$ denote the forward and backward warp intensity differences between template patches and warped images, $\mathbf{u}^f_i$ and $\mathbf{u}^b_j$ denote the estimated forward and backward displacements of patches, and normalization

$$Z = \sum_{i}^{N^f_s} \lambda^f_{i,\mathbf{x}} / \max(1, \|d^f_i(\mathbf{x})\|_2) + \tag{16}$$

$$\sum_{j}^{N^b_s} \lambda^b_{j,\mathbf{x}} / \max(1, \|d^b_j(\mathbf{x})\|_2). \tag{17}$$

Since displacement estimate $\mathbf{u}^b_j$ is generally not integer, we employ bilinear interpolation for the second term in equation (15).

The densification for the dense backward flow field $\mathbf{U}^b_s$ is computed analogously. After the densification step on each scale, the variational refinement is again performed independently for each direction.

(iii) Robust error norms. Equation (1) in the paper minimizes an L2 norm which is strongly affected by outliers. Since the L1 and the Huber norm are known to be more robust towards outliers, we examined their effect on our algorithm. However, the objective function (1) cannot easily be changed to directly minimize a different error norm. However, in each iteration we can transform the error $\varepsilon = I_{t+1}(\mathbf{x}+\mathbf{u}) - T(\mathbf{x})$ for each pixel, such that, implicitly after squaring the transformed error, eq. (1) minimizes a different norm. We transform the error ε on each pixel at location $\mathbf{x}$ as follows:

- L1-Norm:
$$\varepsilon \leftarrow \text{sign}(\varepsilon) \cdot \sqrt{|\varepsilon|}$$
- Huber-Norm:
$$\varepsilon \leftarrow \begin{cases} \varepsilon & \text{, if } \varepsilon < b \\ \text{sign}(\varepsilon) \cdot \sqrt{2b|\varepsilon| - b^2} & \text{, otherwise} \end{cases}$$

After the transformed error ε is squared in the objection function the corresponding L1 or Huber norm is minimized in each iteration.

We plotted the result for all three extensions in Fig. 10, where we compare all extensions on the Sintel training benchmark against our method without these extensions. As baseline we start from operating point **(3)**, as described in the paper, and evaluate all extensions separately. The six numbered operating points correspond to: (1) Baseline, (2) baseline with color images, (3) baseline enforcing forward-backward consistency, (4) baseline using the L1 norm as objective function, (5) baseline using the Huber norm as objective function, (6) baseline using color images, enforcing forward-backward consistency and using the Huber norm as objective function.

We conclude that while all extensions decrease the resulting optical flow error, overall, the obtained error reduction remains too low to justify inclusion of these extensions in the method.

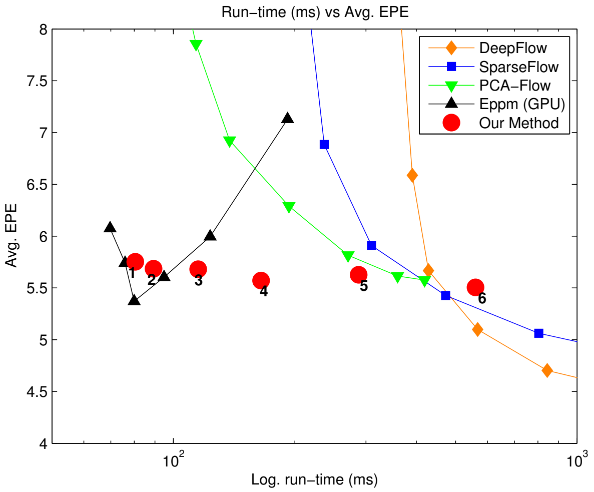


Figure 10: Evaluation of extensions. (1) Baseline, (2) baseline with color images, (3) baseline enforcing forward-backward consistency, (4) baseline using the L1 norm as objective function, (5) baseline using the Huber norm as objective function, (6) baseline using color images, enforcing forward-backward consistency and using the Huber norm as objective function.

C. Implementation details, parallelization and memory consumption

Our method is implemented in C++ and all run-times where computed on a Intel Core i7 3770K CPU at 3.50GHz. In this chapter we provide more engineering details and exact timings of the most important components, details on possible parallelization using OpenMP and overall memory consumption.

Implementation details

For our method no special data structures are needed. What enables the high speed is a combination of a very fast flow initialization of DIS with a slow variational refinement per scale. Initialization and variational refinement constitute 32 % and 57 %, respectively, of the total run-time on each scale. Care was taken to allocate all memory at once, use SSE instructions and inplace-operations whenever possible, and terminate iterations early for small residuals. Bottlenecks for further speed-ups are repeated (bilinearly interpolated) patch extraction (step 3 in **Alg. 1** of the paper) and pixel-wise refinements (step 5).

In Table 5 we show a direct run-time comparison on Sintel- and VGA-resolution images. The run-time scales linearly with the image area.

We will break down the run-time of **3.32 ms** for **DIS (2)** on Sintel-resolution images, running over 3 scales, for all components in table 6.

	DIS **(1)**	**DIS (2)**	DIS **(3)**	DIS **(4)**
Sintel	1.65 / 606	3.32 / **301**	97.8 / 10.2	1925 / 0.52
VGA	0.93 / 1075	2.35 / **426**	70.3 / 14.2	1280 / 0.78

Table 5: Sintel (1024×436), VGA (640×480) run-times in (ms/Hz).

Total run-time (3.32 ms)	ms	% of total run-time
Memory allocation	0.17	5.27
Processing scale $\theta_s = 5$	0.23	6.91
Processing scale $\theta_s = 4$	0.65	19.6
Processing scale $\theta_s = 3$	2.26	68.1

Table 6: Break down of **DIS (2)** total run-time on Sintel images

Run-time grows approximately by a factor of 4 for each finer scale, and is spend similarly on each scale. Representatively for the last scale ($\theta_s = 3$), corresponding to a downscaling factor of 8 and 448 uniformly distributed 8x8 patches, the time of 2.26 ms spend on this layer breaks down following **Alg. 1** in table 7.

Run-time on scale $\theta_s = 3$ (2.26 ms)	ms	% of run-time
Patch initialization, steps (1./2.)	0.09	4.1
Inverse Search, step (3.)	0.74	32.8
Densification, (4.)	0.14	5.9
Variational refinement, (5.)	1.29	57.1

Table 7: Break down of **DIS (2)** run-time on one scale

Step (3.) and (5.) are most time-consuming. Step (3.) breaks down in 1.66 μs for each of 448 patches. This breaks down into 0.08 μs for initialization (gradient/intensity extraction in reference image, computation of Hessian) and 0.13 μs for each of 12 optimization iterations (template intensity extraction, parameter update). Step (5.) breaks down for each of $\theta_{vo} = 4$ iterations into 0.12 ms for computation of the data and smoothness terms and 0.2 ms for solving of the linear systems and updating the flow field per pixel.

Parallelization

We examined the potential speed-up by parallelization using OpenMP on 4 cores. We parallelized step 3 and 5 of our implementation, to operate independently on each patch and pixel, respectively. The run-times for all for operating points are tabulated in Table 8. Since thread creation and management leads to an overhead of a few milliseconds, the speed-up of 4 cores only becomes significant for longer run-times. For longer sequences, where threads are created only once, this overhead would be negligible. Since each tread executes the same instructions and there is no need to communicate, a massive parallelization on a GPU will potentially yield an even larger speed-up.

	DIS **(1.)**	**DIS (2.)**	DIS **(3.)**	DIS **(4.)**	Farneback	PCAFlow	DeepFlow
Peak Mem. (MB)	35.52	**35.56**	100.1	311.9	43.31	1216	6851
Speed (ms)	1.65	**3.32**	97.8	1925	26.27	419.1	55752
Speed (Hz)	606	**301**	10.2	0.52	38.06	2.38	0.018

Table 9: Peak memory consumption (MB) on Sintel-resolution images for all four operating points chosen in the paper and for the baselines: Farneback method, PCAFlow and DeepFlow at full resolutions.

DIS operating point	**(1.)**	**(2.)**	**(3.)**	**(4.)**
Speed-up (x)	1.29	**1.75**	2.15	3.70

Table 8: Speed-up factor from parallelization (OpenMP, 4 cores) for all four operating points chosen in the paper

Memory Consumption

We also examined the peak memory consumption of our algorithm on images from the Sintel benchmark. We tabulated the result in Table 9. Roughly 15 MB are used by image data, and the rest by all patches and associated data, which is pre-allocated during initialization. If parallelization is not used, and memory is a bottleneck, the memory footprint can be reduced by not pre-allocating all patch memory.

D. Plots for more operating points on the KITTI and Sintel benchmarks

We plot the optical flow result on the Sintel and KITTI training benchmarks in Fig. 11. Besides the four operating points chosen in the paper, we plot a variety of operating points for different parameter settings with and without variational refinement. In addition to the observations detailed in the paper, we note that the variational refinement brings the strongest advantage in the small displacement range, whereas large displacement errors cannot be recovered easily by this refinement.

E. Plots for experiments § 3.3 and § 3.4 including preprocessing time

In our evaluation we excluded preprocessing (disk access, image rescaling, gradient computation) for all methods, by carefully timing time spend on these tasks within their provided code. For EPPM, where only binaries where provided, we subtracted the average preprocessing overhead of our method, since we were unable to measure overhead-time directly within their code.

The exclusion of the preprocessing time enables us to compare the actual time spend on flow computation, without evaluating the constant preprocessing overheads. This is particularly important when this preprocessing is shared between tasks or even unnecessary in robotics streaming applications, or when it heavily dominates the run-time and would therefore clutter the analysis.

This is illustrated in Fig. 12, which shows the end-point error on the training sets of Sintel and Kitti (Figures 4 and 5 in the paper in § 3.3, 3.4) versus total run-time of each method *including* preprocessing. It is easy to observe, that in the range of several tens or hundreds of Hertz preprocessing dominates and makes an analysis diffult.

F. More exemplary results for KITTI and Sintel (training)

More exemplary optical flow results on the training subsets of the Sintel [14] and KITTI [21] benchmarks are shown in Fig. 13, 14, 15, and 16. Error maps for Figs. 13 and 14 are plotted in Figs. 17 and 18.

G. Error maps on Sintel-training

Optical flow error maps on the training subset of the Sintel [14] are shown in Fig. 17 and Fig. 18. More error maps on the test sets of Sintel and KITTI can be found online through their test websites. Typical error modes of our method are observable at motion discontinuities (Fig. 17, first block), large displacements (Fig. 17, third block) and at frame boundaries for fast motions (e.g. Fig. 18, second block)

H. More exemplary results for the high frame-rate experiment on Sintel-training in § 3.5

More exemplary results for our high frame-rate experiment (§ 3.5) are shown in Fig. 19 and 20.

I. Depth from Stereo with DIS

We can also apply our algorithm to the problem of computing depth from a stereo pair of images. If the image planes of both cameras are parallel and aligned in depth, the epipoles are at infinity, and the depth computation task becomes the problem of pixel-wise estimation of horizontal displacements. We remove the vertical degree of freedom from our method, and evaluate on the Middlebury dataset for depth from stereo [42]. We evaluate against four methods: Semi-Global Matching (SGM) [26], Block Matching (BM) [30], Efficient Large-scale Stereo Matching (ELAS) [22] and Slanted-Plane Stereo (SPS) [56]. We use the same 4 operating points as in the paper, with one change: iteration numbers are halved, $\theta_{it} \leftarrow \theta_{it}/2$.

The result is displayed in Fig. 21. We have two observations. Firstly, while *operating point* **(1)** and **(2)** are still much faster than all baseline methods for the same error, the speed-benefit is smaller than in the optical flow experiments. Secondly, *for all baseline methods* the optimal performance is achieved with a downscaled input image pair instead of the finest resolution. This suggests that these methods were fine-tuned for images with smaller resolutions than those provided in the recently published Middlebury benchmark. In consequence, for these methods several tuning parameters have to be adapted to deal with large input resolutions. We observe that our method is more robust to those changes.

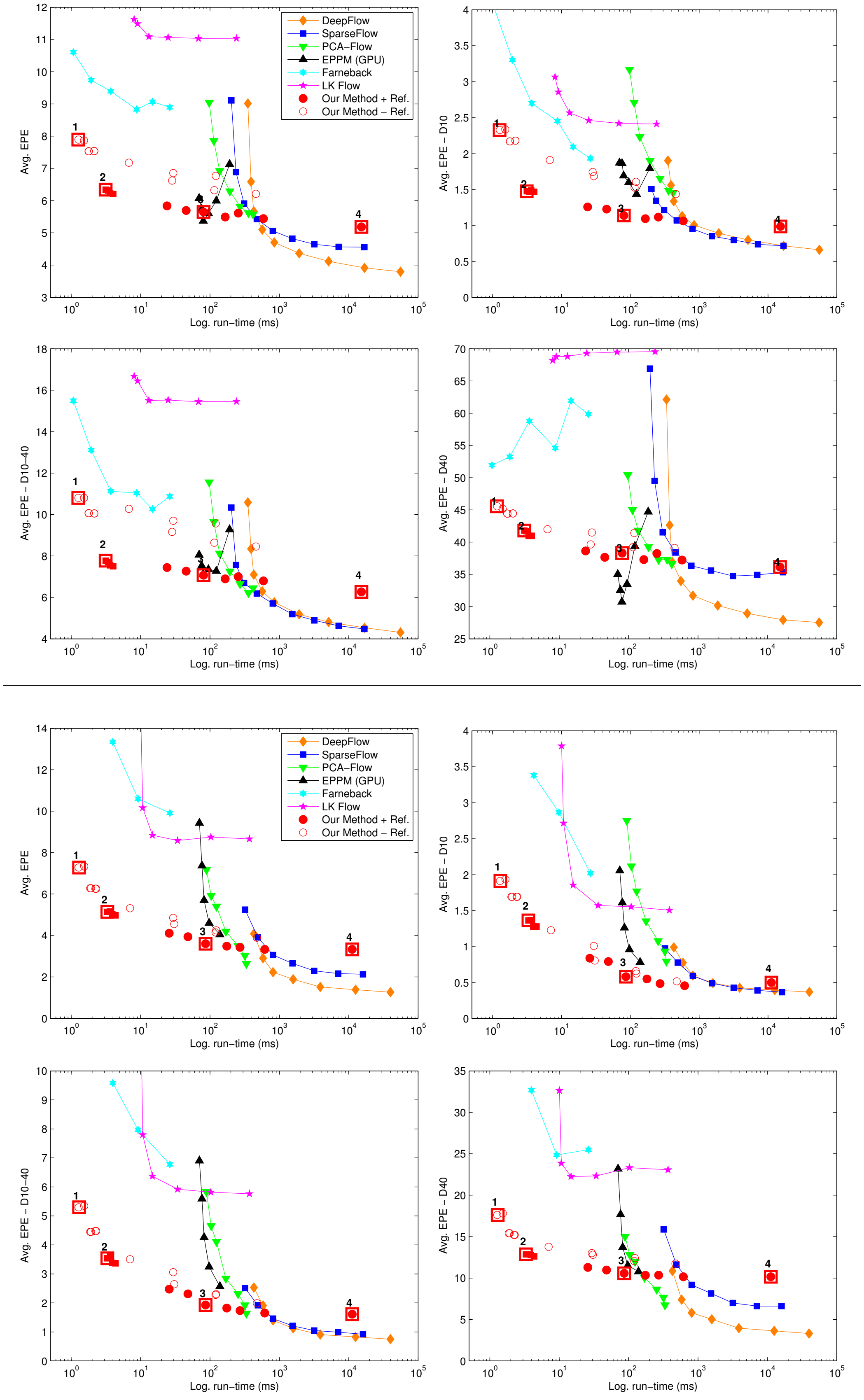


Figure 11: Sintel (top 2 × 2 block) and KITTI (bottom 2 × 2 block) flow result. In each block of 2 × 2, top to bottom, left to right: Average end-point error over image full domain, average end-point error over pixels with ground truth displacement over $<$ 10 pixels, average end-point error over pixels with ground truth displacement between 10 and 40 pixels, average end-point error over pixels with ground truth displacement over $<$ 40 pixels. The four operating points used in the paper are marked with **1**-**4** in the plots.

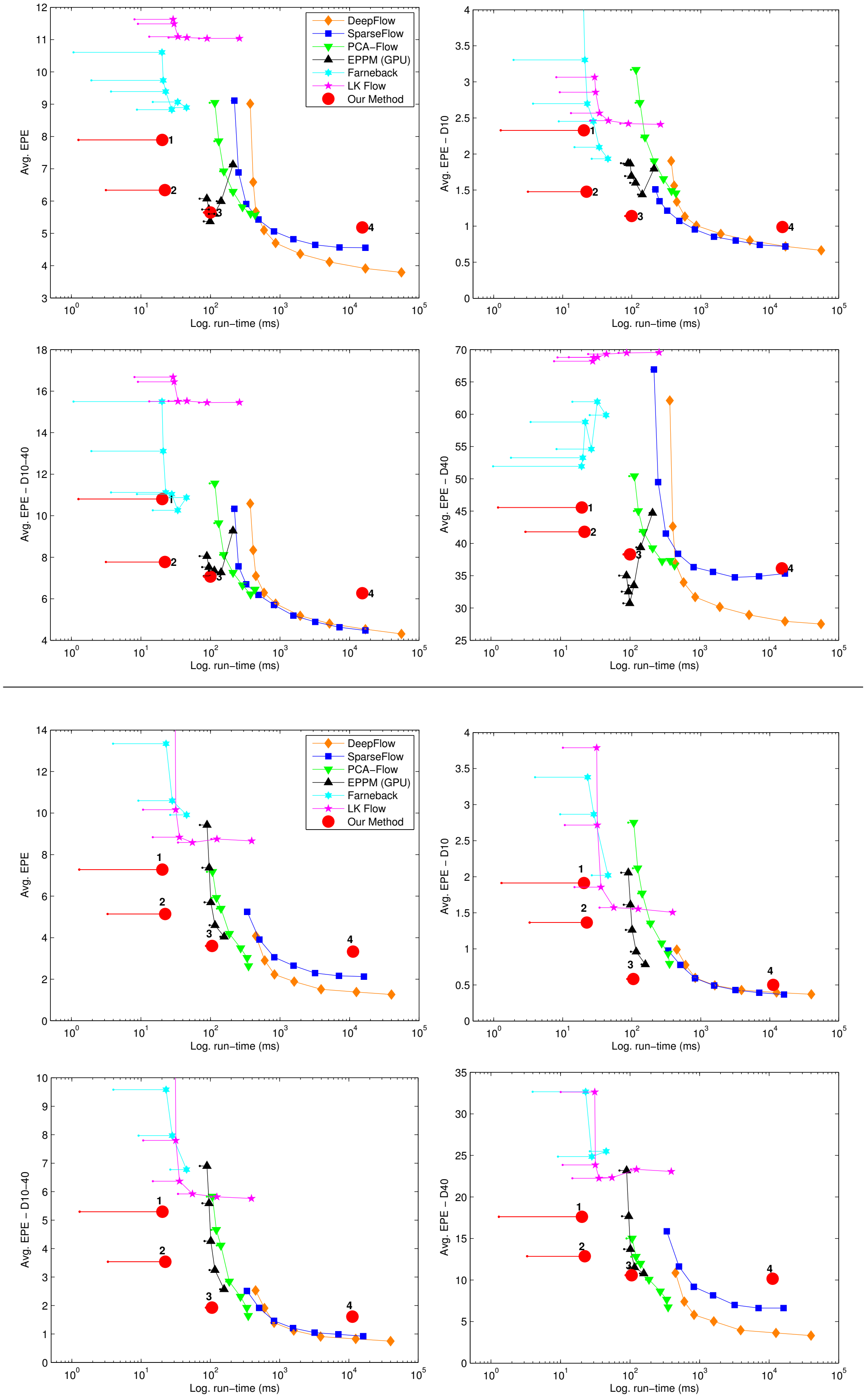


Figure 12: Sintel (top 2 × 2 block) and KITTI (bottom 2 × 2 block) flow result. Same plots as Figure 4 and 5 in the paper, but including preprocessing time for all methods. The horizontal bars for each method indicate portion of total run-time spend in preprocessing (disk access, image rescaling, gradient computation).

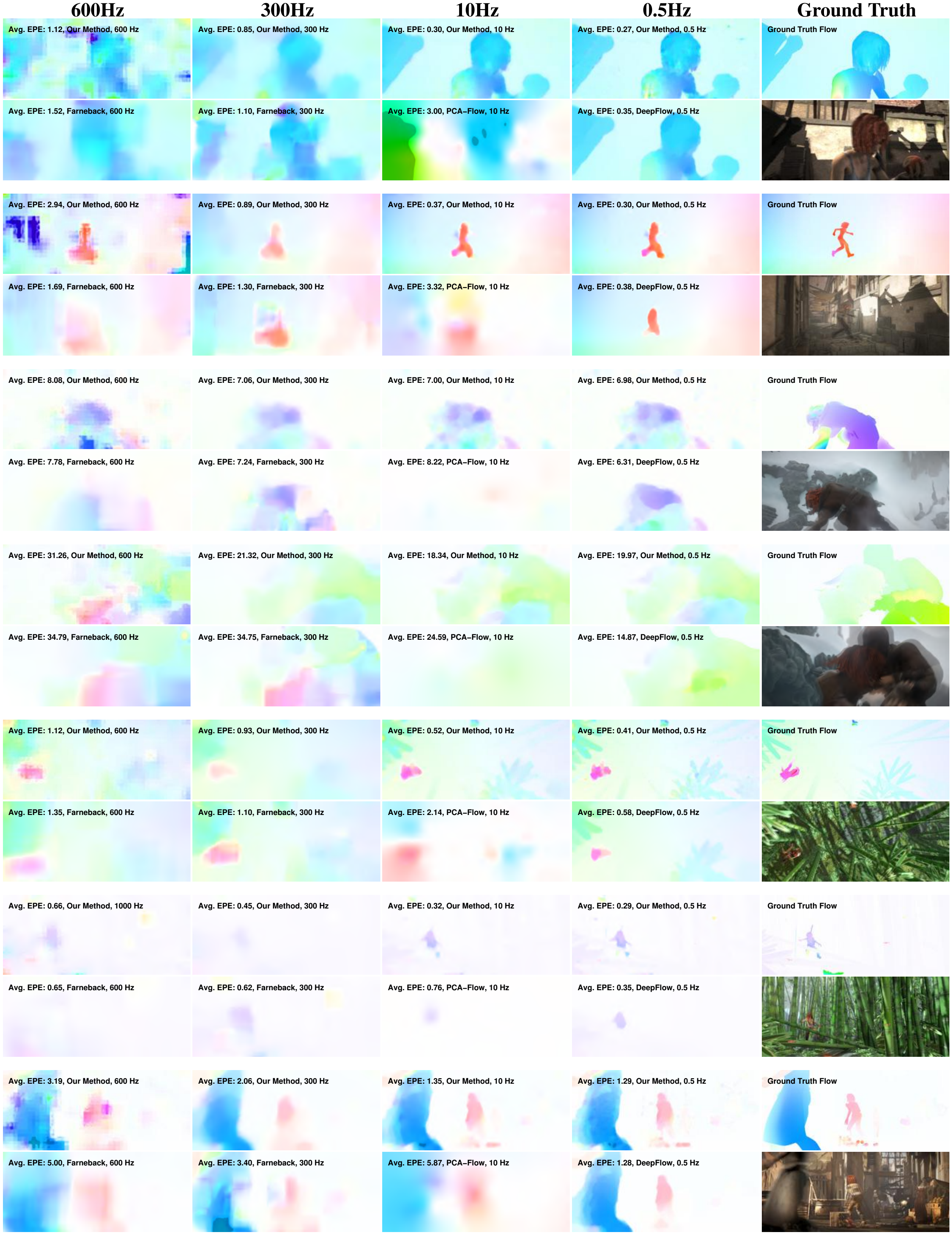

600Hz
300Hz
10Hz
0.5Hz
Ground Truth
Avg. EPE: 1.12, Our Method, 600 Hz
Avg. EPE: 0.85, Our Method, 300 Hz
Avg. EPE: 0.30, Our Method, 10 Hz
Avg. EPE: 0.27, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 1.52, Farneback, 600 Hz
Avg. EPE: 1.10, Farneback, 300 Hz
Avg. EPE: 3.00, PCA–Flow, 10 Hz
Avg. EPE: 0.35, DeepFlow, 0.5 Hz
Avg. EPE: 2.94, Our Method, 600 Hz
Avg. EPE: 0.89, Our Method, 300 Hz
Avg. EPE: 0.37, Our Method, 10 Hz
Avg. EPE: 0.30, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 1.69, Farneback, 600 Hz
Avg. EPE: 1.30, Farneback, 300 Hz
Avg. EPE: 3.32, PCA–Flow, 10 Hz
Avg. EPE: 0.38, DeepFlow, 0.5 Hz
Avg. EPE: 8.08, Our Method, 600 Hz
Avg. EPE: 7.06, Our Method, 300 Hz
Avg. EPE: 7.00, Our Method, 10 Hz
Avg. EPE: 6.98, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 7.78, Farneback, 600 Hz
Avg. EPE: 7.24, Farneback, 300 Hz
Avg. EPE: 8.22, PCA–Flow, 10 Hz
Avg. EPE: 6.31, DeepFlow, 0.5 Hz
Avg. EPE: 31.26, Our Method, 600 Hz
Avg. EPE: 21.32, Our Method, 300 Hz
Avg. EPE: 18.34, Our Method, 10 Hz
Avg. EPE: 19.97, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 34.79, Farneback, 600 Hz
Avg. EPE: 34.75, Farneback, 300 Hz
Avg. EPE: 24.59, PCA–Flow, 10 Hz
Avg. EPE: 14.87, DeepFlow, 0.5 Hz
Avg. EPE: 1.12, Our Method, 600 Hz
Avg. EPE: 0.93, Our Method, 300 Hz
Avg. EPE: 0.52, Our Method, 10 Hz
Avg. EPE: 0.41, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 1.35, Farneback, 600 Hz
Avg. EPE: 1.10, Farneback, 300 Hz
Avg. EPE: 2.14, PCA–Flow, 10 Hz
Avg. EPE: 0.58, DeepFlow, 0.5 Hz
Avg. EPE: 0.66, Our Method, 1000 Hz
Avg. EPE: 0.45, Our Method, 300 Hz
Avg. EPE: 0.32, Our Method, 10 Hz
Avg. EPE: 0.29, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 0.65, Farneback, 600 Hz
Avg. EPE: 0.62, Farneback, 300 Hz
Avg. EPE: 0.76, PCA–Flow, 10 Hz
Avg. EPE: 0.35, DeepFlow, 0.5 Hz
Avg. EPE: 3.19, Our Method, 600 Hz
Avg. EPE: 2.06, Our Method, 300 Hz
Avg. EPE: 1.35, Our Method, 10 Hz
Avg. EPE: 1.29, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 5.00, Farneback, 600 Hz
Avg. EPE: 3.40, Farneback, 300 Hz
Avg. EPE: 5.87, PCA–Flow, 10 Hz
Avg. EPE: 1.28, DeepFlow, 0.5 Hz

Figure 13: Exemplary results on Sintel (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback 600Hz, Farneback 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image. See Fig. 17 for error maps.

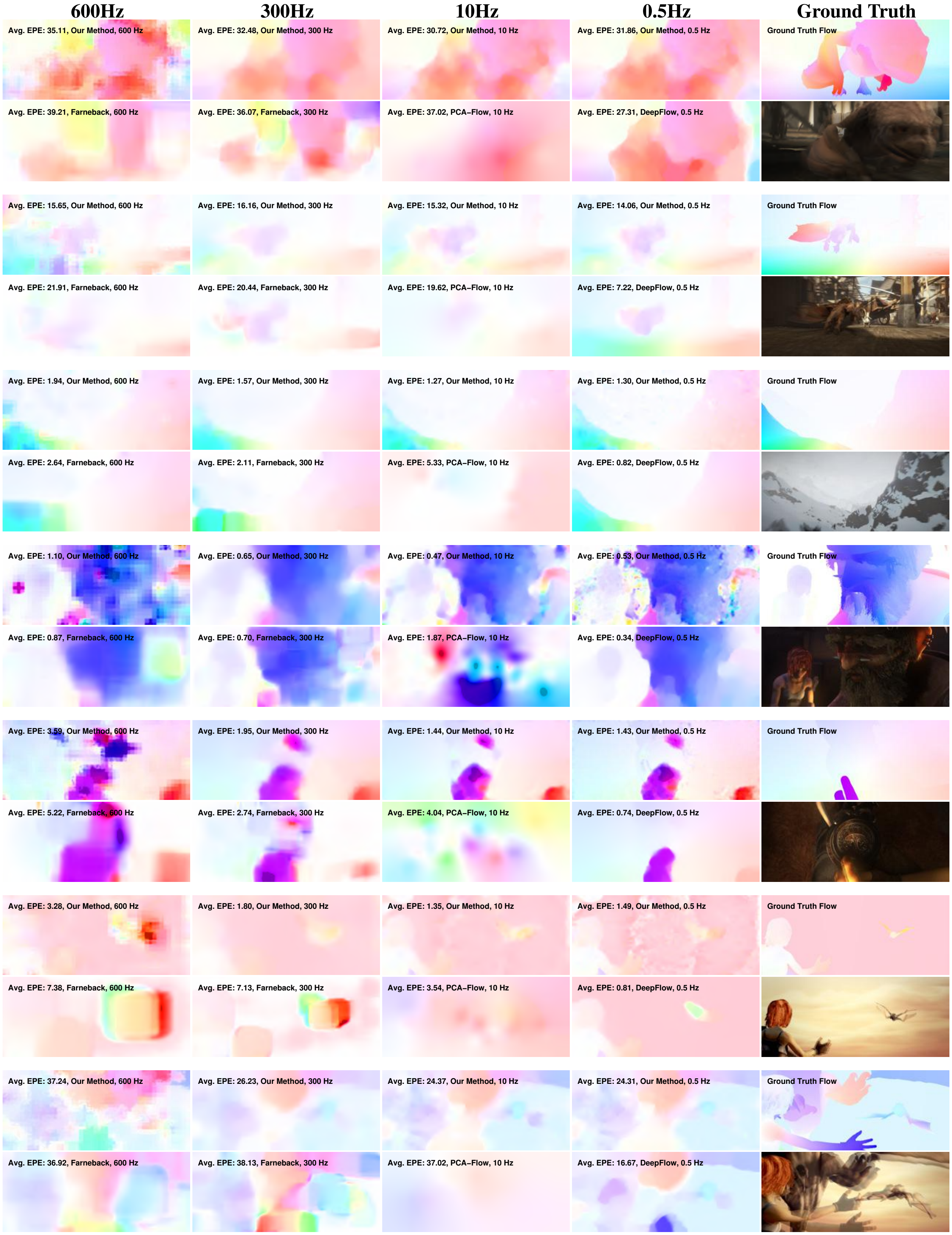


Figure 14: Exemplary results on Sintel (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback 600Hz, Farneback 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image. See Fig. 18 for error maps.

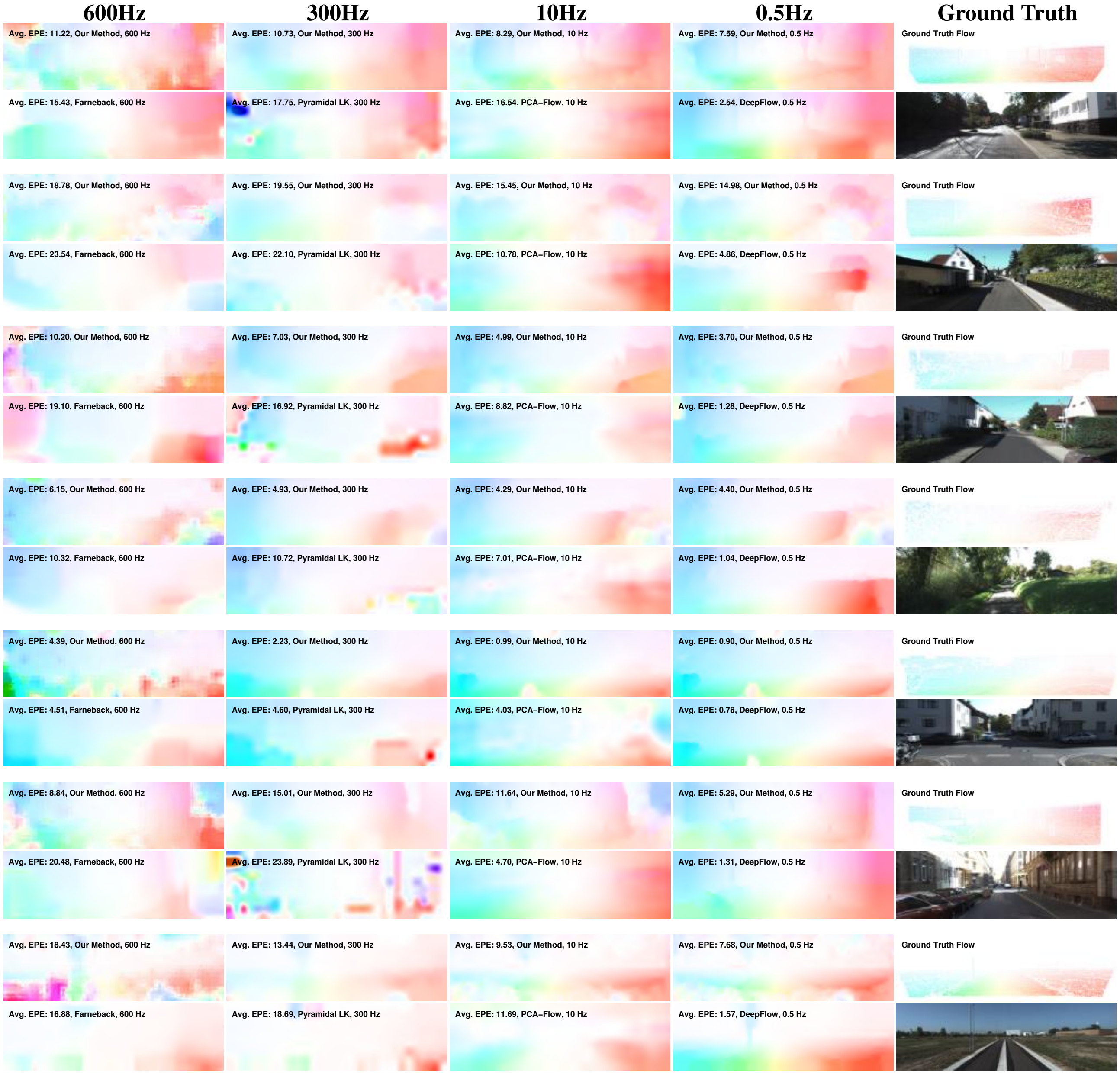

600Hz
300Hz
10Hz
0.5Hz
Ground Truth
Avg. EPE: 11.22, Our Method, 600 Hz
Avg. EPE: 10.73, Our Method, 300 Hz
Avg. EPE: 8.29, Our Method, 10 Hz
Avg. EPE: 7.59, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 15.43, Farneback, 600 Hz
Avg. EPE: 17.75, Pyramidal LK, 300 Hz
Avg. EPE: 16.54, PCA–Flow, 10 Hz
Avg. EPE: 2.54, DeepFlow, 0.5 Hz
Avg. EPE: 18.78, Our Method, 600 Hz
Avg. EPE: 19.55, Our Method, 300 Hz
Avg. EPE: 15.45, Our Method, 10 Hz
Avg. EPE: 14.98, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 23.54, Farneback, 600 Hz
Avg. EPE: 22.10, Pyramidal LK, 300 Hz
Avg. EPE: 10.78, PCA–Flow, 10 Hz
Avg. EPE: 4.86, DeepFlow, 0.5 Hz
Avg. EPE: 10.20, Our Method, 600 Hz
Avg. EPE: 7.03, Our Method, 300 Hz
Avg. EPE: 4.99, Our Method, 10 Hz
Avg. EPE: 3.70, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 19.10, Farneback, 600 Hz
Avg. EPE: 16.92, Pyramidal LK, 300 Hz
Avg. EPE: 8.82, PCA–Flow, 10 Hz
Avg. EPE: 1.28, DeepFlow, 0.5 Hz
Avg. EPE: 6.15, Our Method, 600 Hz
Avg. EPE: 4.93, Our Method, 300 Hz
Avg. EPE: 4.29, Our Method, 10 Hz
Avg. EPE: 4.40, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 10.32, Farneback, 600 Hz
Avg. EPE: 10.72, Pyramidal LK, 300 Hz
Avg. EPE: 7.01, PCA–Flow, 10 Hz
Avg. EPE: 1.04, DeepFlow, 0.5 Hz
Avg. EPE: 4.39, Our Method, 600 Hz
Avg. EPE: 2.23, Our Method, 300 Hz
Avg. EPE: 0.99, Our Method, 10 Hz
Avg. EPE: 0.90, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 4.51, Farneback, 600 Hz
Avg. EPE: 4.60, Pyramidal LK, 300 Hz
Avg. EPE: 4.03, PCA–Flow, 10 Hz
Avg. EPE: 0.78, DeepFlow, 0.5 Hz
Avg. EPE: 8.84, Our Method, 600 Hz
Avg. EPE: 15.01, Our Method, 300 Hz
Avg. EPE: 11.64, Our Method, 10 Hz
Avg. EPE: 5.29, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 20.48, Farneback, 600 Hz
Avg. EPE: 23.89, Pyramidal LK, 300 Hz
Avg. EPE: 4.70, PCA–Flow, 10 Hz
Avg. EPE: 1.31, DeepFlow, 0.5 Hz
Avg. EPE: 18.43, Our Method, 600 Hz
Avg. EPE: 13.44, Our Method, 300 Hz
Avg. EPE: 9.53, Our Method, 10 Hz
Avg. EPE: 7.68, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 16.88, Farneback, 600 Hz
Avg. EPE: 18.69, Pyramidal LK, 300 Hz
Avg. EPE: 11.69, PCA–Flow, 10 Hz
Avg. EPE: 1.57, DeepFlow, 0.5 Hz

Figure 15: Exemplary results on KITTI (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback 600Hz, Pyramidal LK 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image.

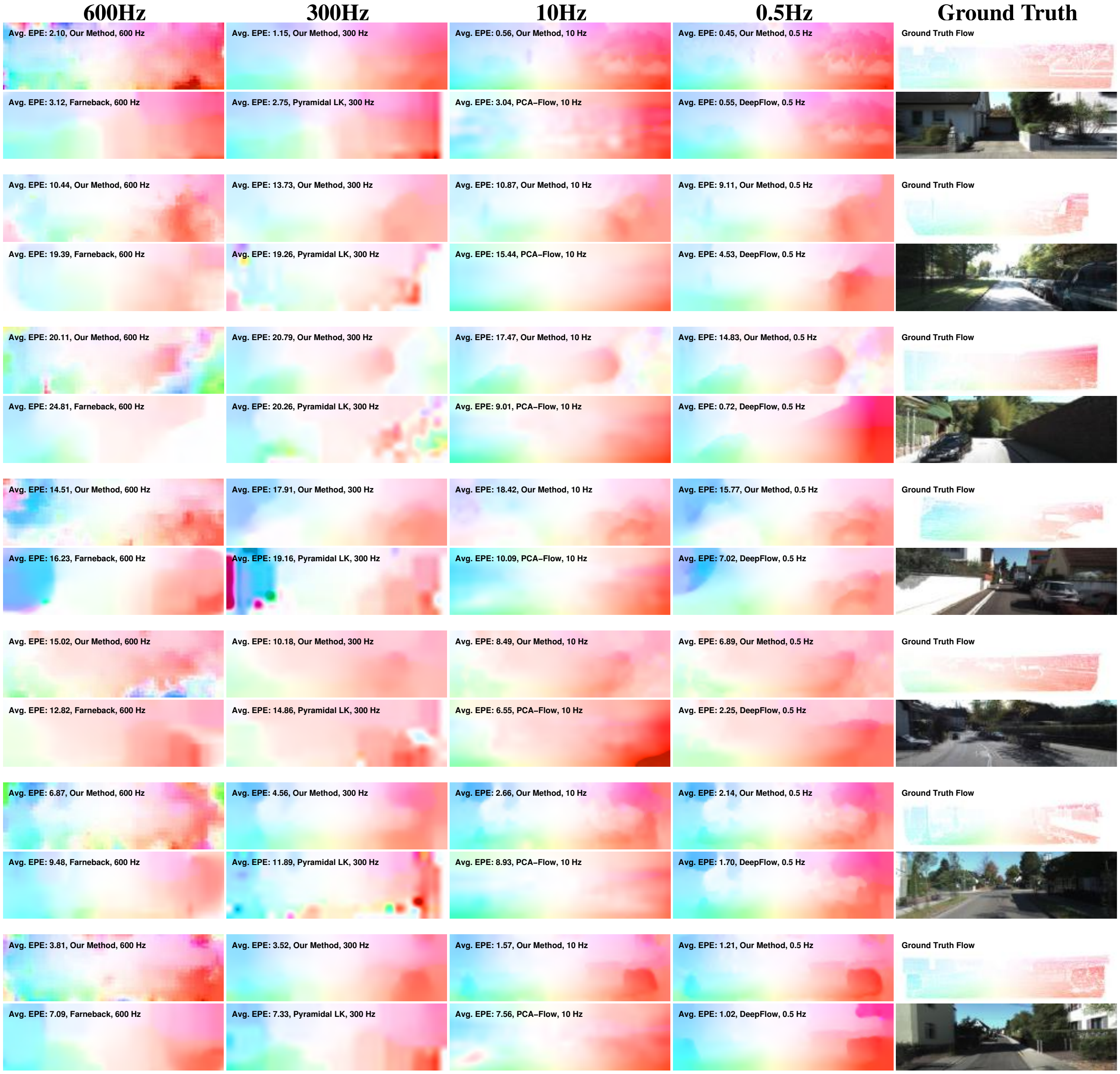

600Hz
300Hz
10Hz
0.5Hz
Ground Truth
Avg. EPE: 2.10, Our Method, 600 Hz
Avg. EPE: 1.15, Our Method, 300 Hz
Avg. EPE: 0.56, Our Method, 10 Hz
Avg. EPE: 0.45, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 3.12, Farneback, 600 Hz
Avg. EPE: 2.75, Pyramidal LK, 300 Hz
Avg. EPE: 3.04, PCA–Flow, 10 Hz
Avg. EPE: 0.55, DeepFlow, 0.5 Hz
Avg. EPE: 10.44, Our Method, 600 Hz
Avg. EPE: 13.73, Our Method, 300 Hz
Avg. EPE: 10.87, Our Method, 10 Hz
Avg. EPE: 9.11, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 19.39, Farneback, 600 Hz
Avg. EPE: 19.26, Pyramidal LK, 300 Hz
Avg. EPE: 15.44, PCA–Flow, 10 Hz
Avg. EPE: 4.53, DeepFlow, 0.5 Hz
Avg. EPE: 20.11, Our Method, 600 Hz
Avg. EPE: 20.79, Our Method, 300 Hz
Avg. EPE: 17.47, Our Method, 10 Hz
Avg. EPE: 14.83, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 24.81, Farneback, 600 Hz
Avg. EPE: 20.26, Pyramidal LK, 300 Hz
Avg. EPE: 9.01, PCA–Flow, 10 Hz
Avg. EPE: 0.72, DeepFlow, 0.5 Hz
Avg. EPE: 14.51, Our Method, 600 Hz
Avg. EPE: 17.91, Our Method, 300 Hz
Avg. EPE: 18.42, Our Method, 10 Hz
Avg. EPE: 15.77, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 16.23, Farneback, 600 Hz
Avg. EPE: 19.16, Pyramidal LK, 300 Hz
Avg. EPE: 10.09, PCA–Flow, 10 Hz
Avg. EPE: 7.02, DeepFlow, 0.5 Hz
Avg. EPE: 15.02, Our Method, 600 Hz
Avg. EPE: 10.18, Our Method, 300 Hz
Avg. EPE: 8.49, Our Method, 10 Hz
Avg. EPE: 6.89, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 12.82, Farneback, 600 Hz
Avg. EPE: 14.86, Pyramidal LK, 300 Hz
Avg. EPE: 6.55, PCA–Flow, 10 Hz
Avg. EPE: 2.25, DeepFlow, 0.5 Hz
Avg. EPE: 6.87, Our Method, 600 Hz
Avg. EPE: 4.56, Our Method, 300 Hz
Avg. EPE: 2.66, Our Method, 10 Hz
Avg. EPE: 2.14, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 9.48, Farneback, 600 Hz
Avg. EPE: 11.89, Pyramidal LK, 300 Hz
Avg. EPE: 8.93, PCA–Flow, 10 Hz
Avg. EPE: 1.70, DeepFlow, 0.5 Hz
Avg. EPE: 3.81, Our Method, 600 Hz
Avg. EPE: 3.52, Our Method, 300 Hz
Avg. EPE: 1.57, Our Method, 10 Hz
Avg. EPE: 1.21, Our Method, 0.5 Hz
Ground Truth Flow
Avg. EPE: 7.09, Farneback, 600 Hz
Avg. EPE: 7.33, Pyramidal LK, 300 Hz
Avg. EPE: 7.56, PCA–Flow, 10 Hz
Avg. EPE: 1.02, DeepFlow, 0.5 Hz

Figure 16: Exemplary results on KITTI (training). In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Farneback 600Hz, Pyramidal LK 300Hz, PCA-Flow 10Hz, DeepFlow 0.5Hz, Original Image.

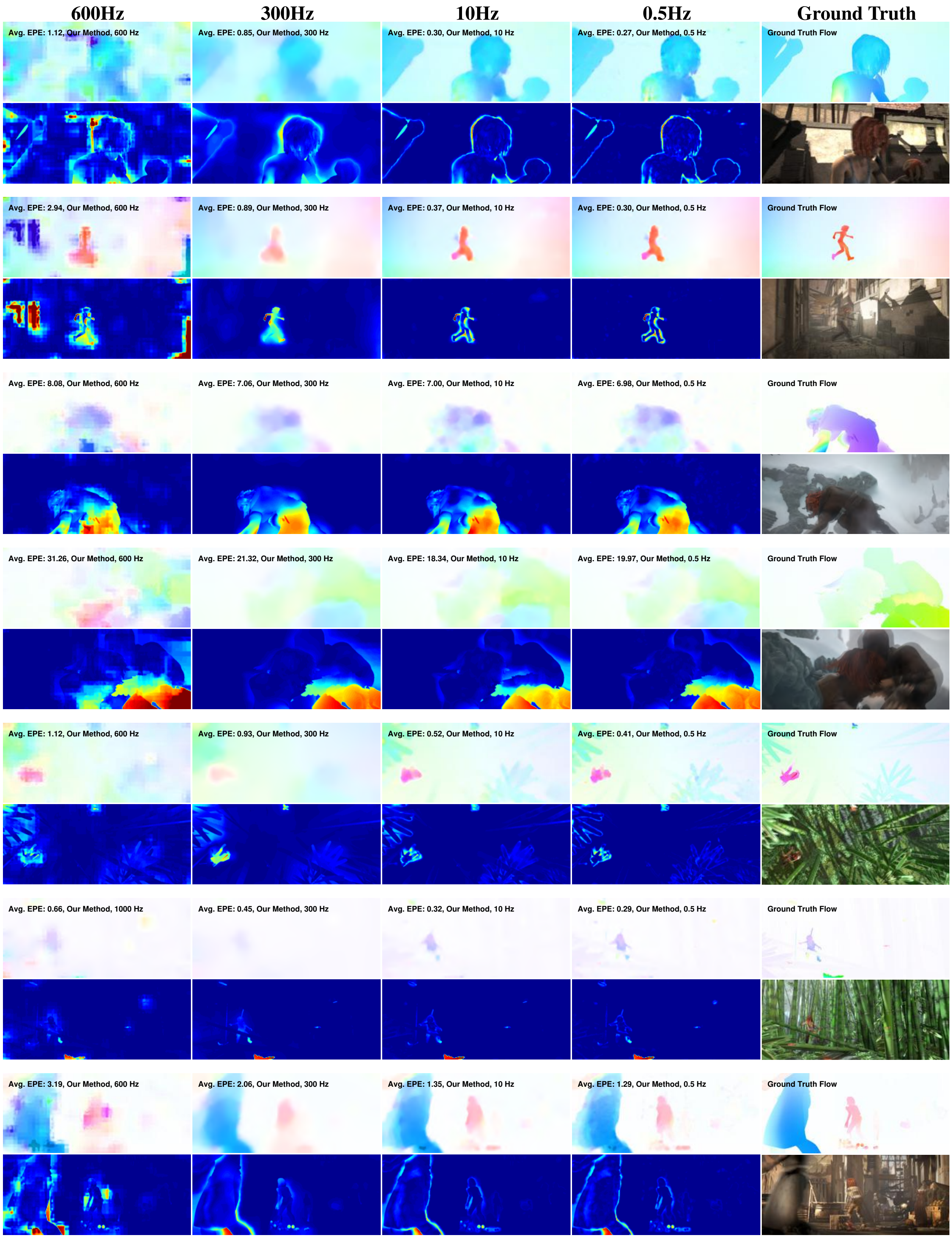


Figure 17: Exemplary results on Sintel (training) and error maps. In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Error heat maps scaled from blue (no error) to red (maximum ground truth flow magnitude), Original Image.

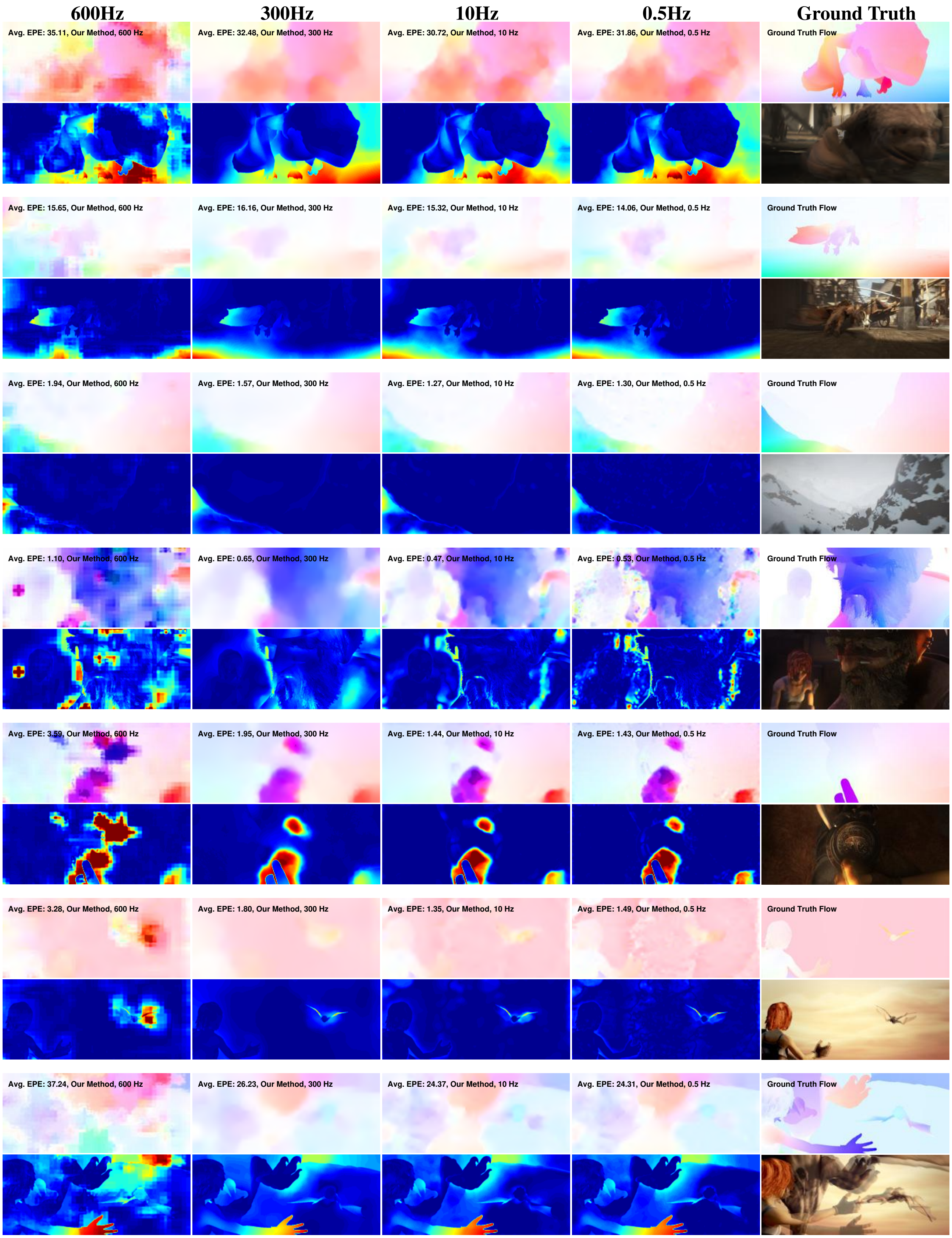


Figure 18: Exemplary results on Sintel (training) and error maps. In each block of 2 × 6 images. Top row, left to right: Our method for operating points (**1**)-(**4**), Ground Truth. Bottom row: Error heat maps scaled from blue (no error) to red (maximum ground truth flow magnitude), Original Image.

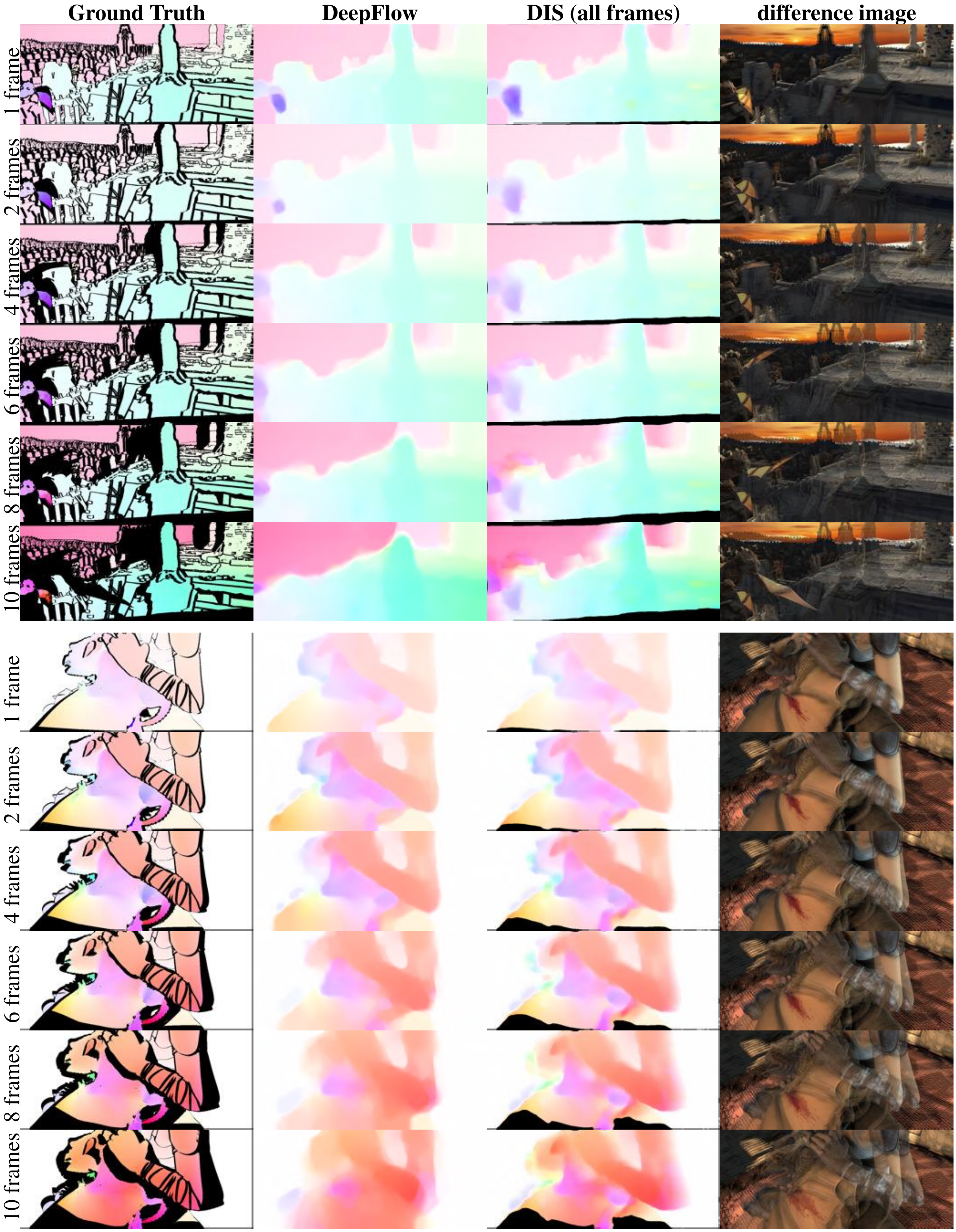


Figure 19: Optical flow on Sintel with lower temporal resolution. In each block of 6x4: Rows, top to bottom, correspond to step sizes 1 (original frame-rate), 2, 4, 6, 8, 10 frames. Columns, left to right, correspond to new ground truth, DeepFlow result, DIS result (through *all intermediate frames*), original images. Large displacements are significantly better preserved by DIS through higher frame-rates.

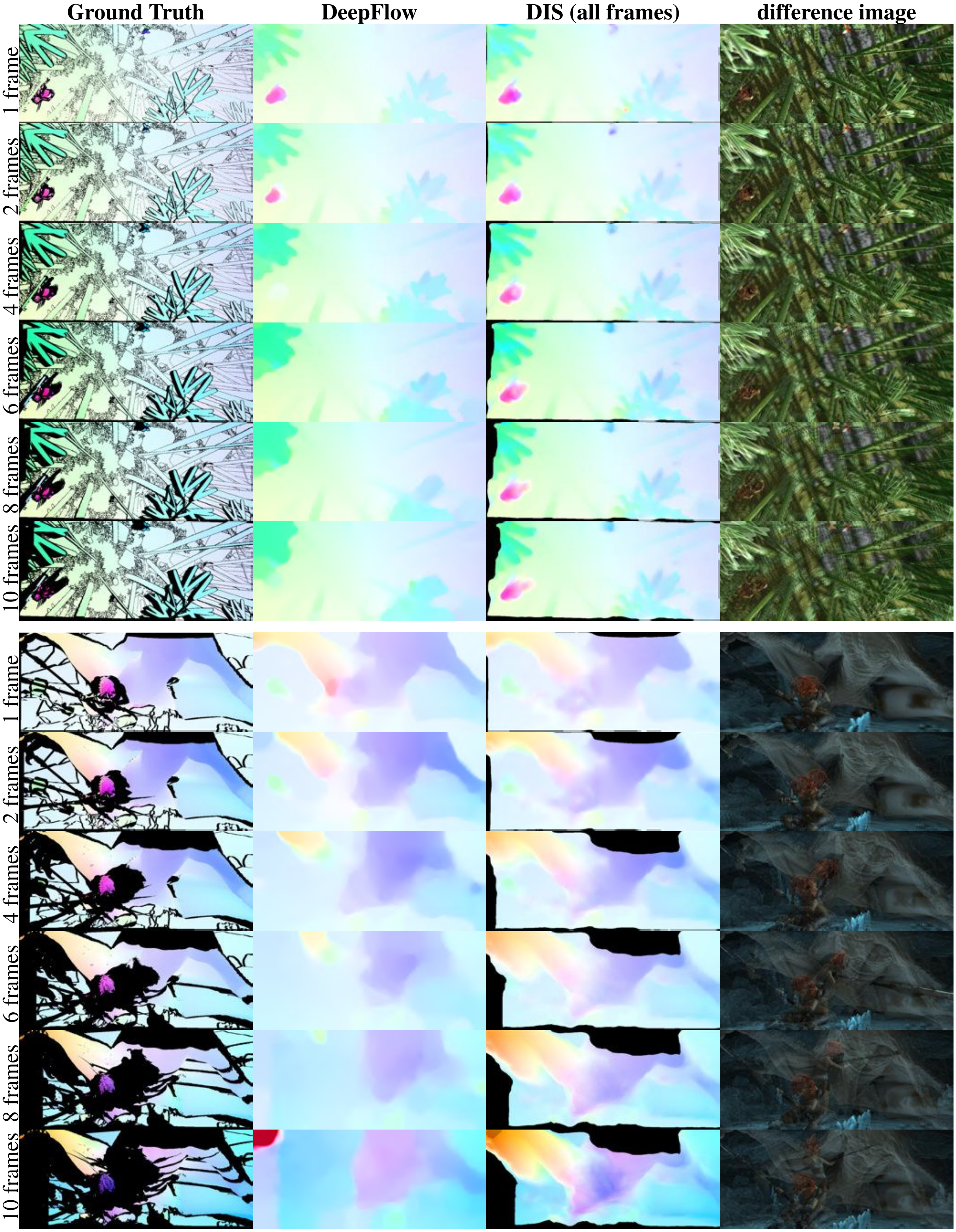


Figure 20: Optical flow on Sintel with lower temporal resolution. In each block of 6x4: Rows, top to bottom, correspond to step sizes 1 (original frame-rate), 2, 4, 6, 8, 10 frames. Columns, left to right, correspond to new ground truth, DeepFlow result, DIS result (through *all intermediate frames*), original images. Large displacements are significantly better preserved by DIS through higher frame-rates.

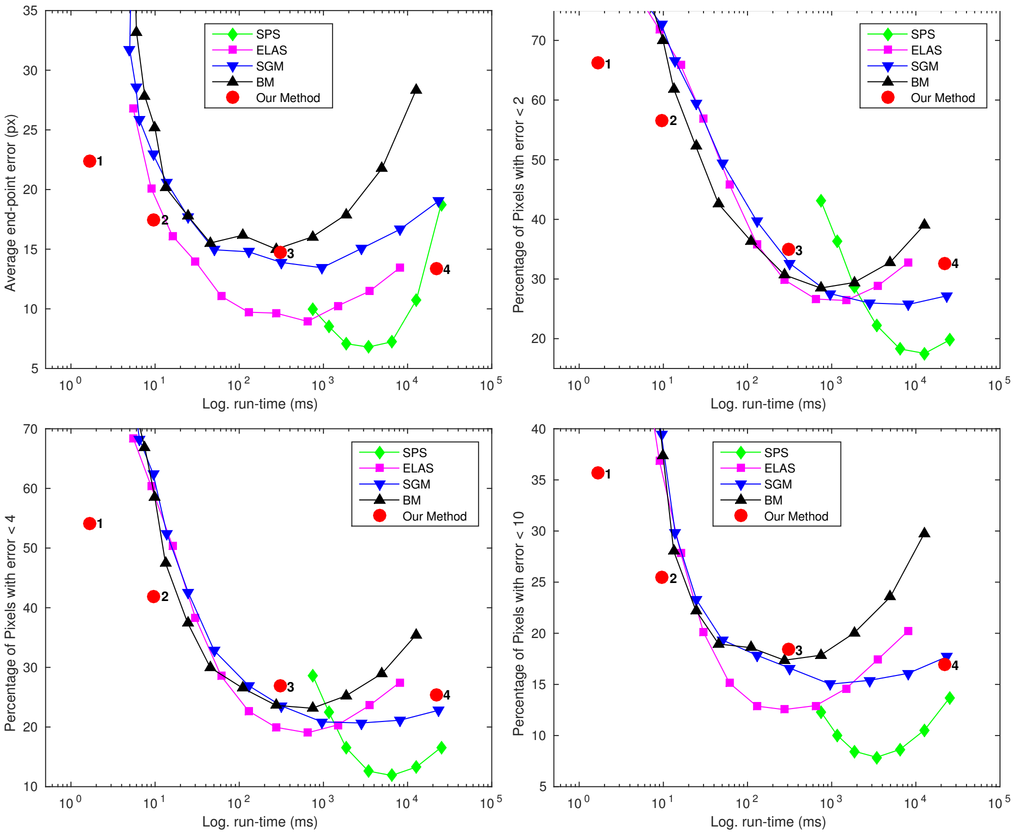


Figure 21: Middlebury depth from stereo results (training dataset). Top Left: Average end-point error (pixel) versus run-time (millisecond), Top Right, Bottom: Percentage of pixels with error below 2, 4 and 10 pixels.